39th International Symposium on Distributed Computing

DISC 2025, October 27–31, 2025, Berlin, Germany

Edited by

Dariusz R. Kowalski



Editors

Dariusz R. Kowalski
School of Computer and Cyber Sciences, Augusta University, GA, USA
dkowalski@augusta.edu

ACM Classification 2012
Software and its engineering → Distributed systems organizing principles; Computing methodologies → Distributed computing methodologies; Computing methodologies → Concurrent computing methodologies; Hardware → Fault tolerance; Information systems → Data structures; Networks; Theory of computation; Theory of computation → Models of computation; Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-402-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-402-4.

Publication date
October, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.DISC.2025.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Dariusz R. Kowalski 0:xi

Organization
..... 0:xiii

Awards and Highlights
..... 0:xvii

2025 Principles of Distributed Computing Doctoral Dissertation Awards
..... 0:xix

2025 Edsger W. Dijkstra Prize in Distributed Computing
..... 0:xxi

Regular Papers

ABEL: Perfect Asynchronous Byzantine Extension from List-Decoding
Ittai Abraham and Gilad Asharov 1:1–1:20

On the Efficiency of Dynamic Transaction Scheduling in Blockchain Sharding
Ramesh Adhikari, Costas Busch, and Miroslav Popovic 2:1–2:23

An Almost-Logarithmic Lower Bound for Leader Election with Bounded Value Contention
Dan Alistarh, Faith Ellen, and Alexander Fedorov 3:1–3:16

pod: An Optimal-Latency, Censorship-Free, and Accountable Generalized Consensus Layer
Orestis Alpos, Bernardo David, Jakov Mitrovski, Odysseas Sofikitis, and Dionysis Zindros 4:1–4:24

DAG It Off: Latency Prefers No Common Coins
Ignacio Amores-Sesar, Viktor Grøndal, Adam Holmgård, and Mads Ottendal 5:1–5:17

Hierarchical Consensus: Scalability Through Optimism and Weak Liveness
Pedro Antonino, Antoine Durand, and A. W. Roscoe 6:1–6:20

On the Shape Containment Problem Within the Amoebot Model with Reconfigurable Circuits
Matthias Artmann, Andreas Padalkin, and Christian Scheideler 7:1–7:22

Auditable Shared Objects: From Registers to Synchronization Primitives
Hagit Attiya, Antonio Fernández Anta, Alessia Milani, Alexandre Rapetti, and Corentin Travers 8:1–8:22

Distributed Download from an External Data Source in Byzantine Majority Settings
John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg 9:1–9:22

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Amnesiac Flooding: Easy to Break, Hard to Escape
Henry Austin, Maximilien Gadouleau, George B. Mertzios, and Amitabh Trehan 10:1–10:23

New Limits on Distributed Quantum Advantage: Dequantizing Linear Programs
Alkida Balliu, Corinna Coupette, Antonio Cruciani, Francesco d'Amore, Massimo Equi, Henrik Lievonen, Augusto Modanese, Dennis Olivetti, and Jukka Suomela 11:1–11:22

Distributed Computation with Local Advice
Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Krzysztof Nowicki, Dennis Olivetti, Eva Rotenberg, and Jukka Suomela 12:1–12:19

Towards Fully Automatic Distributed Lower Bounds
Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Dennis Olivetti, and Joonatan Saarhelo 13:1–13:19

Energy-Efficient Maximal Independent Sets in Radio Networks
Dominick Banasik, Varsha Dani, Fabien Dufoulon, Aayush Gupta, Thomas P. Hayes, and Gopal Pandurangan 14:1–14:24

Asynchronous Latency and Fast Atomic Snapshot
João Paulo Bezerra, Luciano Freitas, Petr Kuznetsov, and Matthieu Rambaud ... 15:1–15:22

Perpetual Exploration in Anonymous Synchronous Networks with a Byzantine Black Hole
Adri Bhattacharya, Pritam Goswami, Evangelos Bampas, and Partha Sarathi Mandal 16:1–16:17

Deterministic Synchronous Self-Stabilizing BFS Construction with Constant Space Complexity
Lélia Blin, Franck Petit, and Sébastien Tixeuil 17:1–17:15

Complexity Landscape for Local Certification
Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun 18:1–18:21

Robust Predicate and Function Computation in Continuous Chemical Reaction Networks
Kim Calabrese, David Doty, and Mina Latifi 19:1–19:23

Two for One, One for All: Deterministic LDC–Based Robust Computation in Congested Clique
Keren Censor-Hillel, Orr Fischer, Ran Gelles, and Pedro Soto 20:1–20:19

Content-Oblivious Leader Election in 2-Edge-Connected Networks
Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou 21:1–21:22

The Complexity Landscape of Dynamic Distributed Subgraph Finding
Yi-Jun Chang, Lyuting Chen, Yanyu Chen, Gopinath Mishra, and Mingyang Yang 22:1–22:20

Boosting Payment Channel Network Liquidity with Topology Optimization and Transaction Selection
Krishnendu Chatterjee, Jan Matyáš Křišťan, Stefan Schmid, Jakub Svoboda, and Michelle Yeo 23:1–23:22

Validity in Network-Agnostic Byzantine Agreement
Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer 24:1–24:23

Model-Agnostic Approximation of Constrained Forest Problems
Corinna Coupette, Alipasha Montaseri, and Christoph Lenzen 25:1–25:24

Towards Constant Time Multi-Call Rumor Spreading on Small-Set Expanders
Emilio Cruciani, Sebastian Forster, and Tijn de Vos 26:1–26:25

On the h-Majority Dynamics with Many Opinions
Francesco d'Amore, Niccolò D'Archivio, George Giakkoupis, and Emanuele Natale 27:1–27:24

Byzantine Consensus in the Random Asynchronous Model
George Danezis, Jovan Komatovic, Lefteris Kokoris-Kogias, Alberto Sonnino, and Igor Zablotchi 28:1–28:22

Strong Linearizability Without Compare&Swap: The Case of Bags
Faith Ellen and Gal Sela 29:1–29:19

Team Formation and Applications
Yuval Emek, Shay Kutten, Ido Rafael, and Gadi Taubenfeld 30:1–30:25

Lower Bounds for k-Set Agreement in Fault-Prone Networks
Pierre Fraigniaud, Minh Hang Nguyen, Ami Paz, Ulrich Schmid, and Hugo Rincon-Galeana 31:1–31:22

Coordination Through Stochastic Channels
Pierre Fraigniaud, Boaz Patt-Shamir, and Sergio Rajsbaum 32:1–32:19

Approach of Agents with Restricted Fuel Tanks
Adam Ganczorz, Tomasz Jurdzinski, Andrzej Pelc, and Grzegorz Stachowiak 33:1–33:22

New Distributed Interactive Proofs for Planarity: A Matter of Left and Right
Yuval Gil and Merav Parter 34:1–34:23

PIPQ: Strict Insert-Optimized Concurrent Priority Queue
Olivia Grimes, Ahmed Hassan, Panagiota Fatourou, and Roberto Palmieri 35:1–35:23

LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics
Martin Hilgendorf and Marina Papatriantafilou 36:1–36:24

Towards Optimal Distributed Edge Coloring with Fewer Colors
Manuel Jakob, Yannic Maus, and Florian Schager 37:1–37:26

Compact Routing Schemes in Undirected and Directed Graphs
Avi Kadria and Liam Roditty 38:1–38:19

TEE Is Not a Healer: Rollback-Resistant Reliable Storage
Sadegh Keshavarzi, Gregory Chockler, and Alexey Gotsman 39:1–39:18

On the Randomized Locality of Matching Problems in Regular Graphs
Seri Khoury, Manish Purohit, Aaron Schild, and Joshua R. Wang 40:1–40:20

Natural Calamities Demand More Rescuers: Exploring Connectivity Time Dynamic Graphs
Ashish Saxena and Kaushik Mondal 41:1–41:23

KUDZU: Fast and Simple High-Throughput BFT
Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen 42:1–42:19

Weight Reduction in Distributed Protocols: New Algorithms and Analysis
Anatoliy Zinovyev 43:1–43:24

Brief Announcements

Brief Announcement: Incrementally Verifiable Distributed Computation
Eden Aldema Tshuva and Rotem Oshman 44:1–44:7

Brief Announcement: DAGs for the Masses
Michael Anoprenko, Andrei Tonkikh, Alexander Spiegelman, and Petr Kuznetsov . 45:1–45:7

Brief Announcement: Communication Patterns for Optimal Resilience
Hagit Attiya, Itay Flam, and Jennifer L. Welch 46:1–46:7

Brief Announcement: Highly Dynamic and Fully Distributed Data Structures
John Augustine, Antonio Cruciani, and Iqra Altaf Gillani 47:1–47:7

Brief Announcement: Distributed Download from an External Data Source in Asynchronous Faulty Settings
John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg 48:1–48:7

Brief Announcement: Synchronization in Anonymous Networks Under Arbitrary Dynamics
Rida Bazzi, Anya Chaturvedi, Andréa W. Richa, and Peter Vargas 49:1–49:8

Brief Announcement: Weaker Assumptions for Asymmetric Trust
Christian Cachin and Juan Villacis 50:1–50:7

Brief Announcement: Non-Uniform Content-Oblivious Leader Election on Oriented Asynchronous Rings
Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou 51:1–51:7

Brief Announcement: From Few to Many Faults: Adaptive Byzantine Agreement with Optimal Communication
Andrei Constantinescu, Marc Dufay, Anton Paramonov, and Roger Wattenhofer . 52:1–52:8

Brief Announcement: Maintaining a Bounded Degree Expander in Dynamic Peer-To-Peer Networks
Antonio Cruciani 53:1–53:7

Brief Announcement: Faster CONGEST Approximation Algorithms for Maximum Weighted Independent Set in Sparse Graphs
Salwa Faour and Fabian Kuhn 54:1–54:7

Brief Announcement: Concurrent Double-Ended Priority Queues
Panagiota Fatourou, Eric Ruppert, and Ioannis Xiradakis 55:1–55:7

Brief Announcement: Universal Dancing by Luminous Robots Under Sequential Schedulers
Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro 56:1–56:7

Brief Announcement: The Virtue of Self-Consistency
Fabian Frei and Koichi Wada 57:1–57:7

Brief Announcement: Optimal-Length Labeling Schemes for Fast Deterministic Communication in Radio Networks
Adam Ganczorz, Tomasz Jurdzinski, and Andrzej Pelc 58:1–58:8

Brief Announcement: Carry the Tail in Consensus Protocols
Suyash Gupta, Dakai Kang, Dahlia Malkhi, and Mohammad Sadoghi 59:1–59:7

Brief Announcement: Distributed Sparsest Cut via Eigenvalue Estimation
Yannic Maus and Tijn de Vos 60:1–60:7

Brief Announcement: Asynchronous Approximate Agreement with Quadratic Communication
Mose Mizrahi Erbes and Roger Wattenhofer 61:1–61:7

Brief Announcement: Time, Fences and the Ordering of Events in TSO
Raïssa Nataf and Yoram Moses 62:1–62:7

Brief Announcement: Optimal Dispersion Under Asynchrony
Debasish Pattanayak, Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma 63:1–63:7

Brief Announcement: Proximal Byzantine Agreement: Improved Accuracy for Fault-Tolerant Replicated Datastreams
Roy Shadmon and Owen Arden 64:1–64:8

Brief Announcement: Congested Clique Counting for Local Gibbs Distributions
Joshua Z. Sobel 65:1–65:7

Brief Announcement: Single-Round Broadcast: Impossibility, Feasibility, and More
Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren 66:1–66:7

Preface

This volume is dedicated to the 39th International Symposium on Distributed Computing (DISC 2025), which was held in October 27 – 31, 2025, in Berlin, Germany. More details can be found on the conference web page: `https://www.disc-conference.org/wp/disc2025/`

DISC Conference Overview

The International Symposium on Distributed Computing (DISC) is an international forum on the theory, design, analysis, implementation and application of distributed systems and networks. It is organized in cooperation with the European Association for Theoretical Computer Science (EATCS). The symposium was established 40 years ago, in 1985, as a biannual International Workshop on Distributed Algorithms on Graphs (WDAG).

Statistics

DISC 2025 received 141 regular submissions and 7 brief announcement submissions. The program was selected by a program committee consisting of 42 members. The program committee was assisted by 123 external reviewers. The committee used a relaxed form of double-blind reviewing, where the submissions themselves were anonymous, but authors were permitted to disseminate their work by uploading to online repositories or by giving talks. Each submission was evaluated by at least three reviewers, and final decisions were made during 3 weeks of online discussions, taking into account the rebuttals. At the end, 43 submissions were accepted as regular papers, and 23 were accepted as brief announcements, of which 2 were submitted in this form and 21 are short versions of regular submissions.

Awards

This volume also includes, in dedicated chapters

- the citations for the best paper and best student paper awards at DISC 2025, as well as the list of highlighted papers,
- the citations for the Best Dissertation Awards, which were presented at PODC 2025,
- the citation for the 2025 Edsger W. Dijkstra Prize in Distributed Computing, which was presented at DISC 2025.

Keynote Talks

The keynote talks at DISC 2025 were given by François Le Gall (Nagoya University, Japan), Moni Naor (Weizmann Institute of Science, Israel – on behalf of the winners of the 2025 Dijkstra Prize), and Ittai Abraham (Intel Labs, Israel).

Acknowledgements

I would like to thank everyone who contributed to DISC 2025, including the authors who submitted their work to DISC 2025, the keynote speakers, all the participants, as well as the whole organizing team listed in the next chapter (Program/Organizing/Steering Committees, external reviewers, sponsors and supporters).

October 2025

Dariusz R. Kowalski
DISC 2025 Program Chair

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee Chair

Dariusz Kowalski	Augusta University, USA

Program Committee

Alkida Balliu	Gran Sasso Science Institute, Italy
Silvia Bonomi	Sapienza University of Rome, Italy
Sebastian Brandt	CISPA, Germany
Costas Busch	Augusta University, USA
Gregory Chockler	University of Surrey, UK
Gianluca De Marco	University of Salerno, Italy
Shlomi Dolev	Ben-Gurion University, Israel
Yuval Emek	Technion, Israel
Antonio Fernandez Anta	IMDEA Software Institute, Spain
Paweł Garncarek	University of Wroclaw, Poland
Ran Gelles	Bar-Ilan University, Israel
Chryssis Georgiou	University of Cyprus, Cyprus
Alexey Gotsman	IMDEA Software Institute, Spain
Rachid Guerraoui	EPFL, Switzerland
Damien Imbs	Aix-Marseille University, France
Marek Klonowski	Wroclaw University of Science and Technology, Poland
Alex Kogan	Oracle Labs, USA
Shay Kutten	Technion, Israel
Jakub Łącki	Google Research, USA
Yannic Maus	TU Graz, Austria
Othon Michail	University of Liverpool, UK
Avery Miller	University of Manitoba, Canada
William K. Moses Jr.	Durham University, UK
Junya Nakamura	Toyohashi University of Technology, Japan
Ruslan Nikolaev	Penn State, USA
Harumichi Nishimura	Nagoya University, Japan
Jan Olkowski	University of Maryland, USA
Dominik Pająk	Google, Poland
Gopal Pandurangan	University of Houston, USA
Boaz Patt-Shamir	Tel Aviv University, Israel
Matthieu Perrin	University of Nantes, France
Maria Potop-Butucaru	Sorbonne University, France
Tomasz Radzik	Kings College London, UK
Joel Rybicki	Humboldt University of Berlin, Germany
Thomas Sauerwald	Cambridge University, UK
Gokarna Sharma	Kent State University, USA
Igor Walukiewicz	CNRS, University of Bordeaux, France
Philipp Woelfel	University of Calgary, Canada
Xiaoming Sun	Chinese Academy of Science, China
Michelle Yeo	National University of Singapore, Singapore
Luca Zanolini	Ethereum, UK

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organizing Committee

Stefan Schmid	TU Berlin, Germany (co-Chair)
Joel Rybicki	HU Berlin, Germany (co-Chair)
Yannic Maus	TU Graz, Austria (Workshops Chair)
William K. Moses Jr.	Durham University, UK (Publicity Chair)

Steering Committee

Hagit Attiya	Technion, Israel (Chair)
Pierre Fraigniaud	CNRS and Université Paris Cité, France (Vice Chair)
Rotem Oshman	Tel Aviv University, Israel (2023 PC Chair)
Dan Alistarh	IST, Austria (2024 PC Chair)
Dariusz Kowalski	Augusta University, USA (2025 PC Chair)
Seth Gilbert	NUS, Singapore (Treasurer)
Yannic Maus	TU Graz, Austria (Member-at-large)

External Reviewers

Ramesh Adhikari	Yarden Adi	Vitaly Aksenov
Timothé Albouy	Abdullah Almethen	Orestis Alpos
Ignacio Amores-Sesar	John Augustine	Mirza Ahad Baig
Leonid Barenboim	Malte Baumecker	Dante Bencivenga
Dominik Bojko	Sébastien Bouchard	Thomas Boudier
Yann Bourreau	Liam Brinker	Trevor Brown
Keren Censor-Hillel	Martin Ceresa	Jérémie Chalopin
Yi-Jun Chang	Soumyottam Chatterjee	Vinicius Cogo
Francesco d'Amore	Peter Davies-Peck	Joshua Daymude
Giuseppe Antonio Di Luna	Gabriel Marques Domingues	Fabien Dufoulon
Faith Ellen	Constantin Enea	Leah Epstein
Laurent Feuilloley	Orr Fischer	Matthias Fitzi
Nathan Flaherty	Cody Freitag	Juan Garay
Thorsten Götte	Diana Ghinea	George Giakkoupis
Yuval Gil	Neil Giridharan	Emmanuel Godard
Wojciech Golab	Barun Gorain	Karol Gotfryd
Christoph Grunau	Homa Habashi	Kun He
David Ilcinkas	Manuel Jakob	Tomasz Jurdziński
Dominik Kaaser	Kritika Kashyap	Sadegh Keshavarzi
Ralf Klasing	David Kozhaya	Piotr Krysta
Ajay Kshemkalyani	Duc Le	Qian Li
Thomas Locher	Giuliano Losa	Maël Luce
Alessia Milani	Anshuman Misra	Masayuki Miyamoto
Anisur Rahaman Molla	Yoram Moses	Miguel A. Mosteiro
Alejandro Naser-Pastoriza	Connor Neibert	Mikhail Nesterenko
Alexandre Nolin	Thomas Nowak	Lutz Oettershagen
Andreas Padalkin	Fragkiskos Papadopoulos	Debasish Pattanayak
Ami Paz	Andrzej Pelc	Asaf Petruschka
Théo Pierron	Antonio Fernández Pinto	Pavan Poudel

Subhajit Pramanick	Pedro Ramalhete	Mikhail Raskin
Raaghav Ravishankar	Peter Robinson	Will Rosenbaum
Ron Rothblum	Tobias Rubel	Florian Schager
Sinchan Sengupta	Nobutaka Shimizu	B Srivathsan
Olivier Stietel	Frederick Stock	Dubois Swan
Adrian Tadic	Francois Taiani	Jhonatan Tavori
David Tench	Corentin Travers	Volker Turau
Jara Uitto	Robin Vacus	Alexandre Vigny
Zhaohui Wei	Francois-Xavier Wicht	Masafumi Yamashita
Sravya Yandamuri	Minji Yang	Penghui Yao
Yitong Yin	Luca Zanetti	Kasra Zardouyan
Jialin Zhang	Jiadong Zhu	Isabella Ziccardi

Acknowledgements

DISC 2025 was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 570230213. We also wish to thank Google for their financial support.

DISC 2025 acknowledges the use of HotCRP for handling submissions and managing the review process, and LIPIcs for producing and publishing the proceedings.



DISC is organized in cooperation with the European Association for Theoretical Computer Science (EATCS).

Awards and Highlights

The Program Committee of DISC 2025 has selected the following paper to receive the **DISC 2025 Best Paper Award**:

Complexity landscape for local certification
by Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun.

One of the most intensively studied topics in distributed graph algorithms in recent years has been to determine the complexity landscape of LCL problems, culminating in an almost complete classification of the asymptotic *time* complexities that LCL problems can exhibit, on both general (bounded-degree) graphs and more restricted graph classes such as paths or trees. The work awarded the DISC 2025 best paper award initiates the study of a distributed *space* complexity landscape, more precisely, the landscape of certificate space complexity in local certification. Besides opening the door to a possibly similarly fruitful line of research as for the aforementioned time complexity landscape, the paper comes with a variety of insightful and surprising results, charting considerable parts of the unknown landscape regimes in different settings. These results range from complexity gap results, showing that no graph property exhibits an optimal certificate space complexity contained in the respectively considered gap, to existence results proving that, for certain space complexities, the properties exhibiting such space complexities exist. The proofs of these results are obtained via an automata-theoretic approach that features a number of novel and clean ideas and differs significantly from the usage of automata theory in the study of the time complexity of LCL problems. The paper also provides several interesting open questions, highlighting the potential this line of research has to offer.

The Program Committee of DISC 2025 has selected the following papers to share the **DISC 2025 Best Student Paper Award**:

Content-Oblivious Leader Election in 2-Edge-Connected Networks
by Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou
and
pod: An Optimal-Latency, Censorship-Free, and Accountable Generalized Consensus Layer
by Orestis Alpos, Bernardo David, Jakov Mitrovski, Odysseas Sofikitis, and Dionysis Zindros.

Lyuting Chen and Haoran Zhou (first paper) and Jakov Mitrovski (second paper) are the student co-authors receiving the shared award.

The paper *"Content-Oblivious Leader Election in 2-Edge-Connected Networks"* makes two fundamental contributions to the design of content-oblivious leader election algorithms over asynchronous networks. The first result is a (non-uniform) deterministic leader election algorithm for general 2-edge-connected graphs, thereby enabling the simulation of arbitrary distributed algorithms without a preselected leader in this model. The second result is a uniform, quiescently terminating deterministic leader election algorithm for unoriented rings, showing that this task is possible even when the ring is unoriented and without requiring additional knowledge. These two results represent a significant advancement in understanding leader election algorithms in content-oblivious asynchronous networks, conjectured to be impossible in the general case (i.e., for uniform algorithms over any 2-edge-connected graph).

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The paper *"pod: An Optimal-Latency, Censorship-Free, and Accountable Generalized Consensus Layer"* introduces Pod, a novel consensus primitive designed to achieve optimal latency, specifically targeting one network round-trip for transaction confirmation. Unlike traditional consensus protocols that depend on inter-replica communication and total-order guarantees, Pod emphasizes low latency, censorship resistance, and accountability. A key safety guarantee is the past-perfection property, which gives clients a stable view of the past: once a round becomes past-perfect, no new confirmed transactions can appear before it. Thus, pod trades off traditional agreement properties in favor of optimal latency, while maintaining sufficient security guarantees to support distributed tasks that do not require total order – such as auctions and payment systems.

The Program Committee has also decided to **highlight** the following papers, selected across major areas of DISC's scope:

- "Team Formation and Applications", by Yuval Emek, Shay Kutten, Ido Rafael, and Gadi Taubenfeld
- "Approach of Agents with Restricted Fuel Tanks", by Adam Ganczorz, Tomasz Jurdzinski, Andrzej Pelc, and Grzegorz Stachowiak
- "Two for One, One for All: Deterministic LDC-based Robust Computation in Congested Clique", by Keren Censor-Hillel, Orr Fischer, Ran Gelles, and Pedro Soto
- "Validity in Network-Agnostic Byzantine Agreement", by Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer
- "Distributed Download from an External Data Source in Byzantine Majority Settings", by John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg
- "Strong Linearizability without Compare&Swap: The Case of Bags", by Faith Ellen and Gal Sela
- "Towards Optimal Distributed Edge Coloring with Fewer Colors", by Manuel Jakob, Yannic Maus, and Florian Schager
- "On the Randomized Locality of Matching Problems in Regular Graphs", by Seri Khoury, Manish Purohit, Aaron Schild, and Joshua R. Wang
- "Distributed Computation with Local Advice", by Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Krzysztof Nowicki, Dennis Olivetti, Eva Rotenberg, and Jukka Suomela.

2025 Principles of Distributed Computing Doctoral Dissertation Awards

The committee for the 2025 Principles of Distributed Computing Doctoral Dissertation Award has decided to share the award between two recipients:

Dr. Christoph Grunau, for his dissertation
"Sampling with Certainty in Distributed and Parallel Computations"

Dr. Vaclav Rozhon, for his dissertation
"Local Complexity: New Results and Bridges to Other Fields"

Dr. Grunau and Dr. Rozhon both completed their Doctor of Sciences degrees at ETH Zürich under the supervision of Mohsen Ghaffari and Angelika Steger.

Dr. Grunau's dissertation describes a local rounding mechanism for transforming many randomized distributed graph algorithms into deterministic algorithms with very little loss in efficiency, using only local information. This gives very efficient deterministic algorithms in the LOCAL model for such fundamental problems as maximal independent set, maximal matching, and graph coloring, resolving numerous long-standing open problems. In the context of parallel algorithms, the dissertation provides a parallel mechanism for derandomizing parallel algorithms that rely on Chernoff-like bounds, giving a deterministic parallel algorithm for set balancing that for the first time achieves work bounds close to those long known for sequential algorithms.

Dr. Rozhon's dissertation describes numerous deep results for the LOCAL model. These include an improved version of the author's breakthrough polylogarithmic-time deterministic algorithm for network decomposition; speedup theorems that provide the final step in completely classifying locally-checkable labeling problems on trees and grids; and a surprising connection between local problems on regular trees and the field of descriptive combinatorics. The dissertation is also noteworthy for its detailed and comprehensive survey of recent progress in the LOCAL model both the author and others.

The committee has also chosen to give an Honorable Mention to

Sean Garnet Ovens, for his dissertation
"The Space Complexity of Asynchronous Algorithms from Bounded Size Base Objects"

Dr. Ovens completed his Doctor of Philosophy degree at the University of Toronto under the supervision of Faith Ellen.

In his dissertation, he describes a new combinatorial technique for proving lower bounds on the space complexity of shared-memory objects implemented from base objects of bounded size. This gives nearly-tight lower bounds on the space complexity of a large class of snapshot-like objects and, in conjunction with a novel valency argument, provides strong lower bounds on the space complexity of obstruction-free consensus implemented from bounded-size swap objects. The final chapter of the dissertation contains a detailed analysis of the space complexity of k-set agreement using unbounded-size swap objects, proving new upper and lower bounds for this problem.

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The 2025 Principles of Distributed Computing Doctoral Dissertation Award Committee:

Dan Alistarh, IST Austria
James Aspnes (chair), Yale University
Pierre Fraigniaud, IRIF, Université Paris Cité, CNRS
Cyril Gavoille, Université de Bordeaux
Danny Hendler, Ben-Gurion University
Petr Kuznetsov, INFRES, Telecom Paris, Institut Polytechnique de Paris

2025 Edsger W. Dijkstra Prize in Distributed Computing

The Edsger W. Dijkstra Prize in Distributed Computing is awarded for outstanding papers on the principles of distributed computing, whose significance and impact on the theory or practice of distributed computing have been evident for at least a decade. It is sponsored jointly by the ACM Symposium on Principles of Distributed Computing (PODC) and the EATCS Symposium on Distributed Computing (DISC). The prize is presented annually, with the presentation taking place alternately at PODC and DISC.

The committee decided to award the 2025 Edsger W. Dijkstra Prize in Distributed Computing to

Moni Naor and Larry Stockmeyer (1948–2004)

for their paper:

"What Can Be Computer Locally?"
SIAM Journal on Computing, volume 24, issue 6, pages 1259–1277, 1995
Proc. of the 25th Annual ACM Symp. on Theory of Computing (STOC), 1993

This is arguably one of the most influential papers in the area of distributed graph algorithms; it is the foundation on which our modern understanding of distributed computational complexity and especially locality of graph problems is largely built. While a lot of early work on distributed graph algorithms and locality studied specific individual problems, Naor and Stockmeyer took the first step towards systematically studying distributed computational complexity of entire problem families. To this end, the paper defines the family of locally checkable labeling problems (LCLs). The family of LCLs as defined by Naor and Stockmeyer is broad enough to capture many distributed graph problems of interest, but it is also narrow enough so that it is possible to prove strong theorems that apply to all LCL problems.

In their paper, Naor and Stockmeyer initiated the study of what distributed complexity classes exist for LCL problems. In particular, they showed that there are nontrivial LCL problems with constant locality. They also initiated the the systematic study of meta-computational questions in this area. Further, the paper introduced the use of Ramsey-type arguments in the context of distributed algorithms, and it initiated the study of the role of randomness in distributed graph algorithms.

The work of Naor and Stockmeyer started a research program that is still very actively being pursued today, 30 years later. This is a paper that is deeply influencing what is happening in our research community, and its impact is rapidly finding its way to also other communities, including at least quantum information theory, measurable combinatorics, stochastic processes, neural networks, and game theory. The paper was a key historical milestone in the development of the field of distributed computing, and it is definitely one of the papers that also younger members of our research community should be familiar with.

The 2025 Edsger W. Dijkstra Prize in Distributed Computing Award Committee:

Hagit Attiya, Technion, Israel
Christian Cachin, U. of Bern, Switzerland
Dariusz R. Kowalski, Augusta University, USA
Fabian Kuhn (chair), U. of Freiburg, Germany
Yoram Moses, Technion, Israel
Paul Spirakis, U. of Liverpool, UK

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ABEL: Perfect Asynchronous Byzantine Extension from List-Decoding

Ittai Abraham
Intel Labs, Petah Tikva, Israel

Gilad Asharov
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

Abstract

Asynchronous byzantine agreement extension studies the message complexity of L-bit multivalued asynchronous byzantine agreement given access to a binary asynchronous Byzantine agreement protocol.

We prove that asynchronous byzantine agreement extension can be solved with perfect security and optimal resilience in $O(nL + n^2 \log n)$ total communication (in bits) in addition to a single call to a binary asynchronous Byzantine agreement protocol. For $L = O(n \log n)$, this gives an asymptotically optimal protocol, resolving a question that remained open for nearly two decades.

List decoding is a fundamental concept in theoretical computer science and cryptography, enabling error correction beyond the unique decoding radius and playing a critical role in constructing robust codes, hardness amplification, and secure cryptographic protocols. A key novelty of our perfectly secure and optimally resilient asynchronous byzantine agreement extension protocol is that it uses list decoding - making a striking new connection between list decoding and asynchronous Byzantine agreement.

2012 ACM Subject Classification Security and privacy; Security and privacy → Cryptography; Security and privacy → Information-theoretic techniques; Security and privacy → Distributed systems security

Keywords and phrases Asynchronous Byzantine Agreement, Perfect Security

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.1

Related Version *Full Version*: `https://eprint.iacr.org/2025/1488` [2]

Funding *Gilad Asharov*: Research supported by the Israel Science Foundation (grant No. 2439/20), and by the European Union (ERC, FTRC, 101043243). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

1 Introduction

The *agreement* problem is perhaps the quintessential problem in fault-tolerant distributed computing. In agreement, each party has an input and the goal is for all honest parties to output (*liveness*) the same value (*agreement*), and that if all of them have the same input, then this is the output (*weak validity*). In *Byzantine* agreement, the challenge is to do this while tolerating a strongly adaptive adversary that can corrupt up to t parties. It is known that for perfect security, or in asynchrony, byzantine agreement requires $n > 3t$, so we call a protocol that can tolerate $n = 3t + 1$ *optimally resilient*. In *asynchronous* byzantine agreement the network is asynchronous. In this setting infinite execution must exist, but using randomization, protocols for binary asynchronous byzantine agreement with an expected $O(1)$ rounds and expected $O(n^2)$ communication are known (for example, given access to a weak coin that has a constant probability of success) [18, 5].


© Ittai Abraham and Gilad Asharov;
licensed under Creative Commons License CC-BY 4.0



The $O(n^2)$ cost for agreement on one bit is optimal due to lower bounds of Dolev and Reischuk [12] even in synchrony and for omission failures and even with randomization against a strongly adaptive adversary [6].

For the multivalued case, where the input is L bits, the obvious additional lower bound is $\Omega(nL)$ bits that are required when $\Omega(n)$ parties do not have the output as their input, need to learn the output. Thus, we say that a multivalued asynchronous byzantine agreement has *asymptotically optimal complexity* if the communication complexity is $O(nL + n^2)$. For $L = \Omega(n \log n)$, we say a multivalued asynchronous byzantine agreement that has $O(nL + n^2 \log n)$ communication complexity has *asymptotically optimal complexity.*

The goal of the asynchronous byzantine agreement extension problem, as suggested in [9, 15], is to study the costs of multivalued asynchronous byzantine agreement given access to a binary byzantine agreement protocol.

In this setting, for $L = \Omega(n \log n)$, results with optimal resilience and asymptotically optimal complexity are known under cryptographic assumptions and a PKI [17, 19] (in fact, using a PKI allows for stronger external validity properties). In synchrony, the breakthrough of [7] obtains perfect security (also see improvements [3] and statistical security with fewer rounds [1]).

Recently, the results in the asynchronous model have seen great progress: using just a cryptographic hash function [14]; near optimal resilience and statistical security [13]; and perfect security with $O(\log n)$ overhead in communication and time, or perfect with $n = 5t+1$ [8] or near optimal resilience [13].

Despite nearly two decades and considerable recent work, the following remains open: for $L = \Omega(n \log n)$, does there exist an asynchronous byzantine agreement extension protocol that is:

1. Optimally resilient ($n = 3t + 1$);
2. Asymptotically optimal (for $L = \Omega(n \log n)$ has complexity $O(Ln)$);
3. Perfectly secure (is error free).

The main result of this paper is a positive answer to this open question.

▶ **Theorem 1.1** (Main). *For $L = \Omega(n \log n)$, there exists a protocol that solves multivalued asynchronous byzantine agreement with optimal resilience, asymptotically optimal complexity, and perfect security, given access to a single instance of binary asynchronous byzantine agreement.*

In other words, our protocol achieves $O(nL + n^2 \log n)$ for inputs of size L in addition to a single call to a binary agreement. A key novelty of our perfectly secure and optimally resilient Asynchronous Byzantine Agreement Extension protocol is that it uses **list decoding** - making a striking new connection between list decoding and asynchronous byzantine agreement. List decoding [20, 16] is a fundamental concept in theoretical computer science and cryptography, allowing error correction beyond the unique decoding radius and playing a critical role in many areas of theoretical computer science. In this paper, we show its criticality in enabling optimal asynchronous extension protocols with perfect security.

As a warm-up, we show a variant with statistical security (also see [13]) to highlight some of the challenges before going to the perfect security case.

▶ **Theorem 1.2** (Statistical). *For an error parameter $0 < \epsilon < 1$, and $L = \Omega(n \log n/\epsilon)$, there exists a protocol that solves multivalued asynchronous byzantine agreement with optimal resilience, asymptotically optimal complexity, and statistical security (with error ϵ), given access to a single instance of binary asynchronous byzantine agreement.*

1.1 Discussion and open questions

Our extension is asymptotically optimal when $L = \Omega(n \log n)$ bits. Obtaining asymptotic optimality for $L = O(n)$ remains an open question. Our reduction is nearly quadratic, but the currently best binary byzantine agreement protocol with perfect security, assuming private channels, is $\Omega(n^4)$ [4]. Improving this bound, or finding other assumptions where binary ABA with perfect security is (nearly) quadratic, is another open question.

Just like [7], we obtain byzantine agreement with a weak form of validity that allows parties to agree on a special symbol $\perp$ if the inputs from the honest parties do not agree. Getting similar results for stronger notions of validity (or proving this to be impossible) also remains an open question.

1.2 High level ideas

We start with an overview of the breakthrough COOL protocol of [7] and how it can be used in asynchrony. Then describe a new functionality called BOOST. The next step is a new general extension framework that combines BOOST, COOL, and binary asynchronous byzantine agreement to achieve asynchronous byzantine agreement extension. We then overview BOOST with statistical security, and then BOOST with perfect security.

COOL in asynchrony. A modular view of the COOL protocol [7, 3] is that it is a weak variant of asynchronous verifiable information dissemination and dispersal [10, 11, 3]. In asynchrony, we can combine the two phases to get a *reliable agreement* protocol that we will call *COOL reliable agreement* or simply COOL. Roughly speaking, COOL gives: (Validity) If all honest start with the same input, they all output this value; (Totality) If an honest outputs, then all honest will output; (Agreement) The output of all honest is the same.

Note that COOL reliable agreement does not guarantee termination in all cases. It is safe, but live only in the good case (validity property). Can we use COOL along with a binary asynchronous agreement protocol to get multivalued asynchronous agreement? This is where the BOOST protocol comes in.

BOOST in asynchrony. Unlike COOL, which is safe but not live, BOOST is live but not safe. All parties eventually output a value, or eventually output $\perp$ - but these are not necessarily mutually exclusive. A party might output twice in the protocol: a value, and later also $\perp$.

In BOOST, every party will eventually output a value or output $\perp$ or do both. Roughly speaking, BOOST gives: (Validity) If all honest start with the same input, they all output this value and never output $\perp$; (Value Totality) If an honest outputs a value then all honest will output a value; ($\perp$ Totality) If an honest outputs $\perp$ then all honest will output $\perp$; and finally the correctness property: (Correct or Detect) Either all parties output the same value, or all parties output $\perp$.

In the *correct* case, all parties output the same value from BOOST. But what if they do not? Then we are in the *detect* case: we are guaranteed that all parties will eventually output $\perp$. However, parties do not know what case they are in. They may output a value first and may not be certain if they will ever eventually also output $\perp$.

This leads to the following natural protocol: run BOOST. If it outputs $\perp$ then enter the binary agreement with 0. If BOOST outputs a value, then enter COOL with this value. If COOL outputs a value, then enter the binary agreement with 1. Finally, if the binary agreement ends with 0, then output $\perp$. Otherwise, wait for the output of COOL and output that.

Roughly speaking, (Validity) If all honest have F then BOOST will only output F and COOL will output F and agreement will output 1; (Liveness) If BOOST outputs $\perp$ then all will eventually output $\perp$ so the binary agreement will complete (note, may not always be on 0). Otherwise, if there is no output of $\perp$, then there is agreement (from the correct or detect property in BOOST), so all will output the same and enter COOL with the same and from the validity of COOL and the liveness of the binary agreement, all will output the same value; (Agreement) If the binary outputs 0 then it is immediate and otherwise due to the agreement property of COOL.

Why is it called BOOST? From BOOST validity, if all honest have $F(x)$ and the t malicious have $G(x) \neq F(x)$, then the output must be $F(x)$. Consider a world where $t+1$ honest have $F(x)$, t honest have $G(x)$, and the malicious are silent. This world is indistinguishable from the validity world with t slow parties with $F(x)$ instead of the silent malicious. Hence, BOOST cannot output $\perp$ because of validity, so it must output a value. That means it must output $F(x)$.

So the $t+1$ that hold $F(x)$ must convince the t honest parties that hold $G(x)$ to also output $F(x)$. So we need to **boost** the value $F(x)$ from $t+1$ inputs to the output of all $2t+1$ honest parties. Hence, we call this protocol BOOST because its main challenge is to identify a value that appears in $t+1$ honest parties and **boost** it to all parties (or detect a problem while trying to do that).

BOOST with statistical security. For statistical security, one can test whether two polynomials are equal (with statistical error) by evaluating them at a random point. Extending this idea, a party P_i can determine whether P_j and P_k hold the same polynomial by receiving a single evaluation at a common random challenge point. This approach, used to reduce rounds in reliable agreement for statistical security [1], treats an evaluation as a hash of the polynomial. The only requirement is that the challenge point be chosen after the adversary fixes the polynomial; for simplicity in our statistical case, we assume all inputs are fixed before the protocol begins (See [1] for removing this assumption).

In the following, define the *F-case* as the case there are at least $t+1$ parties that have the same input F.

Statistical BOOST protocol in 7 phases:

1. *Exchange phase*: parties send a random challenge point, and get responses that are evaluations at the challenge point. In this phase parties also detect if they see $t+1$ evaluations that disagree with their input.
2. *Support phase*: If a party sees $t+1$ responses on the same point, it identifies a group of $t+1$ parties with the same input. It sends them a *support* message. Note that you can send support messages for at most two groups (because $3t+3 > 3t+1$).
3. *Sending YourPoint phase*: If a party hears $2t+1$ support messages on its input then it sends each party its point on this polynomial.
   In the F case, if a party has input $G(x) \neq F(x)$ and hears $2t+1$ support, this means that all parties will see at least $t+1$ support for $G(x)$, and hence all $t+1$ parties with input $F(x)$ will detect.
4. *Sending MyPoint phase*: If a party hears $t+1$ parties send it the same YourPoint, then it sends back this point as a MyPoint message. Critically, a party sends only one MyPoint message.
   So in the F-case, if no $G(x) \neq F(x)$ has $2t+1$ support then eventually all honest parties will send their MyPoint for $F(x)$.

5. *Reconstruct phase*: Parties now wait to robustly reconstruct at least $2t + 1$ MyPoint values that agree on the same polynomial.
6. *Totality and Abundance phase*: standard techniques to ensure that one termination implies all terminate and that at least $t + 1$ terminate with a value.
7. *Disseminate phase*: standard techniques to ensure that if at least $t + 1$ have the same value, and all other have this value or $\perp$ then all honest will output this value. This part is done as a separate sub-protocol.

BOOST with perfect security. Moving from statistical to perfect security introduces several new challenges.

First, in the statistical setting, either $t + 1$ evaluations agree on your input or $t + 1$ disagree, and you detect. With perfect security, detecting from $2t + 1$ responses is much harder. Moreover, sending support requires finding a set of $t + 1$ parties that have the same polynomial. How can you do this if you only hear two deterministic evaluation points from each party?

Second, unlike the statistical setting, we cannot get strong detection properties so early in the protocol, hence we cannot stop after sending one MyPoint and we may need to send several MyPoints. But this complicates decoding: how can you decode if each party sends two points? multiple polynomials may fit. Even more problems arise since a support message may help several polynomials, not just one with high probability as in the statistical setting.

Third, is the most subtle, in the F-case we must ensure that if some party outputs $G(x) \neq F(x)$ then all parties will eventually detect. But party i may have $G(x)$ such that $G(i) = F(i)$, so detecting a conflict seems challenging.

To address the first problem, we use a more subtle detection rule (see Claim 5.2). This detection uses *List Decoding*: we observe that identifying $t + 1$ points that agree on a degree $< t/7$ polynomial is exactly a List Decoding instance and prove the list is at most of size 3.

For the second problem, list decoding helps again: instead of interpolating, we wait to see if there are $2t + 1$ points that agree with a polynomial in our list. This may yield a list of reconstructed polynomials, unlike the statistical case. This adds complexity and forces us to do several non-trivial rounds of detection in order to make sure that in the F-case, if a party outputs $G(x) \neq F(x)$ then all parties eventually detect.

Obtaining this detection is the crux of our protocol, and is done in three phases. The *Filter* phase detects one type of disagreement and then the *Simple Detect* phase detects another. After these two phases there may still be a party that wants to output $G(x) \neq F(x)$ but has $G(i) = F(i)$. In this case, we need to add an additional *Resolve Conflict* phase.

This phase again uses list decoding, allowing every party to learn about $F(x)$. So parties holding $G(x)$ with $G(i) = F(i)$ learn that evaluations at i cannot differentiate parties holding $G(x)$ from parties holding $F(x)$, so they send a new challenge point i' such that $G(i') \neq F(i')$ and using responses from this new challenge point we can finally get the desired detection property.

BOOST with perfect security - Overview of the protocol. Recall that the *F-case* is when there are at least $t + 1$ parties that have the same input F.

1. *Exchange phase*: here each party i with input $f_i(x)$ sends to each party j the two points $f_i(i)$ and $f_i(j)$. There are two non-trivial aspects when receiving these points:
   a. The first is that a party detects if the number of parties that send detect plus the number of parties that send points that do not agree with your input is more than $t + 1$. This rule is used in a subtle manner to prove the detect property in Claim 5.2 in the not F-case.

   b. The second is that we need to give support to any set of $t+1$ parties that agree on the same degree at most $d < t/7$ polynomial, but we cannot use a random challenge point for perfect security. So the only way to find this is to use (online) List Decoding to generate a dynamic set S^1. Since our degree is $< t/7$, the list has at most 3 elements (see Claim 5.2).
2. *Filter phase*: each party sends its support to at most 3 polynomials by sending back to them their point on this polynomial. Now a party may see 3 different values from each other party, how can it know if it has a support of 2t+1? The new idea here is to check which of the polynomials in S^1 has $2t+1$ points that agree with it. Parties put those polynomials in a dynamic set S^2.
   Parties also detect if a polynomial other than their input is in S^2. In the F-case, we prove that all parties whose input disagrees with $F(x)$ will eventually detect. Going forward, let $|B|$ be the number of parties detected this way.
   This detection is not enough, because some party may get $G(x) \in \mathsf{S}^2$ with $G(x) \neq F(x)$ but we may still not have $t+1$ detections. The next two phases solve this.
3. *Simple detect phase*: In this phase if any party i adds $G(x) \neq F(x)$ to S^3 and $G(i) \neq F(i)$ then we prove all parties will eventually detect. We do this by proving that at least $t+1-|B|$ parties that have $F(x)$ as input must detect (see Claim 5.4) and use the $|B|$ detections from the previous phase.
4. *Resolve conflict phase*: Here we want to detect if a party i outputs $G(x) \neq F(x)$ but has $G(i) = F(i)$. We observe that if the previous detect does not trigger, then *all* honest parties have points that agree with $F(x)$ (even if the polynomial they have is not $F(x)$). So how can a party holding $G(x) \neq F(x)$ with $G(i) = F(i)$ know that it is not holding F(x) in this case?
   This observation implies that party i can list decode the values it receives at this stage, and it must include $F(x)$ in this list after waiting for $2t+1$ messages. Now party i, that is holding $G(x) \neq F(x)$, is aware of $F(x)$.
   Party i sees that $G(i) = F(i)$ so index i is not suitable for disambiguating between $F(x)$ and $G(x)$. So party i chooses a new index i' such that $G(i') \neq F(i')$ and sends that as a *new challenge point*. With this new challenge point, in the F-case, if a party outputs $G(x) \neq F(x)$ then it must be that all parties output detect (see Claim 5.5). This is somewhat reminiscent of the random point challenge used in the statistical security setting.
5. *Totality and Abundance phase*: standard techniques.
6. *Disseminate phase*: standard techniques.

2 Preliminaries and Building Blocks

2.1 Notations

We let n denote the total number of participants. We let t denote the total number of corrupted parties. We assume Byzantine faults, and that $n \geq 3t+1$. Here we focus on the case where $n = 3t+1$. We assume communication channels are identifiable so parties can identify who sent them a message.

Reed Solomon codes. Let $\mathbb{F}$ be a finite field such that $|\mathbb{F}| > n$, where n is the number of parties. Without loss of generality, we assume that $1, \ldots, n \in \mathbb{F}$, while this is just to ease convention. To encode a message $m = (m_0, \ldots, m_d)$ where each $m_i \in \mathbb{F}$, the encoding algorithm defines a polynomial $P(x) = m_0 + m_1 x + \ldots + m_d x^d$, and outputs $(P(1), \ldots, P(n))$.

Since two polynomials of degree d can agree on at most d points, the distance is $n-d$. Thus, we can correct up to $(n-d-1)/2$ errors. When $d < n/3$, this means that we can correct, in particular, t errors. For a set S of points, we use $\mathsf{RSDec}(S, d)$ as the unique decoding procedure of Reed-Solomon codes, which returns a unique polynomial of degree d that agrees with all but at most d points in S (if exists).

List decoding. List decoding procedure of Reed-Solomon codes allows the return of all polynomials that agree with a set S when the distance is larger than the minimum distance of the code. We use $\mathsf{ListDecode}(S, d, p)$ to denote the procedure that returns all polynomials of degree d that agree with S on p points. Looking ahead, we will use it with $d = t/7$ and $p = t+1$, and our analysis would show that when $2t+1 \leq |S| \leq 3t+1$, there are no more than three possible polynomials that satisfy that, i.e., no more than three polynomials of degree $t/7$ that agree with $t+1$ points on S.

Input length. In our protocols, we assume that the input is a polynomial of degree d (looking ahead, t in the statistical, and $t/7$ in perfect). For the general case of an L-bit message, the parties segment the L-bit message into $L = \lceil \frac{L}{(d+1)\log|\mathbb{F}|} \rceil$ blocks of size $(d+1)\log|\mathbb{F}|$ bits each (with padding when necessary). Generalizing the protocols to L-blocks input can be done in a natural way, e.g., instead of sending a single point, we would send L points, each is associated with a different polynomial. We omit such details. Additionally, in the perfect security case, we assume $\log|\mathbb{F}| \in O(\log n)$.

2.2 Dispersal

▶ **Definition 2.1** (Dispersal). *A protocol for parties $P_1, \ldots, P_n$ where the input of each party P_i is some $f_i(x) \in \mathbb{F}^{d+1}$, is an* asynchronous dispersal protocol tolerating t corrupted parties *if the following properties hold:*

- **Termination:** *If one honest party terminates, then all honest parties terminate.*
- **Weak agreement:** *If an honest party terminates, then at least $t+1$ honest parties terminate with the same output $F(x)$, and the rest might terminate with either $F(x)$ or $\perp$.*
- **Weak Validity:** *If all honest parties start with the same polynomial $F(x)$ of degree at most d, then termination and weak agreement hold with respect to $F(x)$.*

It is important to note that this protocol might never terminate. In particular, termination is guaranteed only in the case where all honest parties have the exact same input. We might also terminate with other outputs, in which case weak agreement is guaranteed ($t+1$ honest parties output the same output, but the rest output $\perp$).

Protocol 2.1: Dispersal

Input: Each party P_i holds $f_i(x)$ of degree at most d over $\mathbb{F}$.
The protocol:

1. Initialization: Each party initializes $S_i = \emptyset$, $\mathsf{A}_i^1 = \mathsf{A}_i^2 = \emptyset$.
2. **Exchange:**
   a. P_i sends $(\texttt{Exchange}, f_i(i), f_i(j))$ to each P_j.
3. **Dynamic set A_i^1:**
   a. **Upon** receiving $(\texttt{Exchange}, u_j, v_j)$ from P_j, if $f_i(j) = u_j$ and $f_i(i) = v_j$ then add j to A_i^1.
   b. **Upon** $|\mathsf{A}_i^1| \geq n - t$, send OK_1 to all.

4. **Dynamic set A_i^2:**
   a. **Upon** receiving message OK_1 from P_j for which $j \in \mathsf{A}_i^1$, then add j to A_j^2.
   b. **Upon** $|\mathsf{A}_i^2| \geq n - t$, send OK_2 to all.
5. **Sending** `Done`:
   a. If P_i sent OK_2, and **upon** receiving $2t + 1$ OK_2 messages, send `Done` messages to all.
   b. **Upon** receiving $t + 1$ `Done` messages from distinct parties, send `Done` to everyone.
   c. **Upon** receiving $2t + 1$ `Done` messages, if P_i sent OK_2 message, then terminate and output $f_i(x)$. If P_i did not send OK_2 message, then terminate and output $\perp$.

The following theorem is proven in [3], and the protocol is based on the protocol of [7]:

▶ **Theorem 2.2** ([3]). *Protocol 2.1 is a perfectly-secure asynchronous dispersal protocol tolerating $t < n/3$ malicious parties (as per Definition 2.1). The protocol takes 4 rounds and a total communication of $O(nL + n^2 \log n)$ bits where each party starts with an input of size L.*

We remark that Protocol 2.1 is described where the input of each party is a polynomial $f_i(x)$ of degree d, whereas the above theorem refers to a general input of size L. See remark about input length in Section 2.1. We also remark that in [3], the protocol assumes that $d < t/3$.

2.3 Asynchronous Data Dissemination

▶ **Definition 2.3.** *A protocol for parties $P_1, \ldots, P_n$ where the input of each party P_i is some $f_i(x) \in \mathbb{F}^{d+1}$ is a* Asynchronous Data Dissemination protocol *tolerating t corrupted parties if the following properties hold:*

- **Termination:** *If one honest party terminates, then all parties terminate.*
- **Validity:** *If at least $t + 1$ honest parties start with the same input $F(x)$, and all other honest parties start with $\perp$, then all honest parties output $F(x)$.*

Protocol 2.2: Asynchronous Data Dissemination

Input: Each party P_i holds $f_i(x)$ as input, of degree at most d. Some P_i might have input $\perp$.

The protocol:

1. Initialize a multi-set $M_i = \emptyset$ and $S_i = \emptyset$. If $f_i(x) \neq \perp$, send to each P_j its point $(\mathsf{YourPoint}, f_i(j))$.
2. **Upon** receiving $(\mathsf{YourPoint}, u_j)$ from P_j, add u_j to M_i.
3. **Upon** some u_i appearing $t + 1$ times in M_i, send $(\mathsf{MyPoint}, u_i)$ to all parties.
4. **Upon** receiving $(\mathsf{MyPoint}, u_j)$ from P_j, add (j, u_j) to S_i. **Upon** $|S_i| \geq d + t + 1$ execute the following:
   a. Run $\mathsf{RSDec}(S_i)$ and try to decode a polynomial $f_i(x)$ of degree at most d that agrees with S_i on at least $d + t + 1$ values.
   b. If no such polynomial exists, then wait to receive more points in S_i and retry.
   c. If such a polynomial $f_i(x)$ is computed, set $f_i(x)$ to be the resultant value.
5. **Upon** unique decoding $f_i(x)$, terminate and output $f_i(x)$.

▶ **Theorem 2.4** ([10, 3]). *Protocol 2.2 is an asynchronous data-dissemination protocol tolerating $t < n/3$ malicious parties. The protocol takes 2 rounds and $O(nL + n^2 \log n)$ communication, where each party starts with an input of size L.*

2.4 Reliable Agreement

In reliable broadcast, there is a sender, and other parties have no inputs. The sender wishes to broadcast its message, and that all honest parties would output that value. Agreement and validity are guaranteed, but there is no guaranteed termination.

We now consider "reliable agreement". The difference from reliable broadcast is that all parties have an input, and there is no sender.

▶ **Definition 2.5.** *A protocol for parties $P_1, \ldots, P_n$, where the input of each party P_i is some $f_i(x) \in \mathbb{F}^{d+1}$ is a* Reliable Agreement *tolerating t corrupted parties, if the following properties hold:*

- **Termination:** *If one honest party terminates, then all parties terminate.*
- **Validity:** *If all honest parties start with the same input $F(x)$, then all honest parties terminate with output $F(x)$.*
- **Agreement:** *If one honest party terminates with output $F'(x)$, then it is guaranteed that all honest parties eventually terminate with output $F'(x)$.*

Protocol 2.3: Reliable Agreement

- **Input:** The input of each party is some polynomial $f_i(x)$ of degree d over $\mathbb{F}$.
- **The protocol:**
  1. Run Dispersal (Protocol 2.1 with input $f_i(x)$.
  2. **Upon** dispersal terminating with output $f_i^{(1)}(x)$, enter dissemination (Protocol 2.2) with that input.
  3. **Upon** dissemination terminating with output $f_i^{(2)}(x)$, output this polynomial.

The following is proven in the appendix:

▶ **Theorem 2.6.** *Protocol 2.3 is a reliable agreement protocol tolerating $t < n/3$ corrupted parties. It requires $O(nL + n^2 \log N)$ communication for an input of size L.*

2.5 Asynchronous Byzantine Agreement

In an asynchronous Byzantine Agreement protocol, each party has some input x_i, and the parties have to agree on some output. It is required that all parties must terminate, must agree on the output, and if all honest parties start with the same input x, then the output x.

▶ **Definition 2.7.** *A protocol for parties $P_1, \ldots, P_n$ where the input of each party P_i is some bit $b_i \in \{0, 1\}$ is a* Asynchronous Byzantine Agreement *tolerating t corrupted parties if the following properties are satisfied:*

- **Agreement:** *All honest parties must output the same value.*
- **Validity:** *If all honest parties enter with the same input x, then all output x.*
- **Termination:** *All honest parties must terminate.*

If $x \in \{0, 1\}$, then we call the protocol Binary Asynchronous Byzantine Agreement*; If the input is a general string, then we call the protocol* Multi-value Asynchronous Byzantine Agreement*. If a Multi-value Asynchronous Byzantine Agreement uses a Binary Asynchronous Byzantine Agreement, then we call it a* Asynchronous Byzantine Agreement Extension *protocol.*

3 From BOOST to Multivalue Byzantine Agreement

3.1 BOOST

We introduce a new primitive called BOOST. We build a multivalue Byzantine Agreement protocol from BOOST, which we define its properties next:

▶ **Definition 3.1.** *A protocol for parties $P_1, \ldots, P_n$, where the input of each party P_i is some polynomial $f_i(x)$ of degree d over $\mathbb{F}$, is a* BOOST *protocol, tolerating t corrupted parties, if the following properties are satisfied:*

- **Syntax:** *The protocol might not terminate, but each party eventually writes to (at least) one of two output tapes –* output_i*, or/and* detect_i*.*
- **Validity:** *If all honest parties enter with the same polynomial $F(x)$, then at least $t+1$ honest parties set their output to $F(x)$. Moreover, no honest party activates output* detect*.*
- **Set output:** *If an honest party sets output then all honest parties set their output.*
- **Detect or correct:** *If at least $t+1$ honest parties hold the same polynomial $F(x)$ as input, then: (a) either eventually all honest parties activate their detect output tape; or (b) at least $t+1$ honest parties set their output tape to $F(x)$, and the remaining set their output to* proceed*.*
- **Detect:** *If there is no set of $t+1$ honest parties with the same input, then all honest parties activate their detect tape.*

In fact, the crux is in designing this primitive. In the next subsection, we show how to utilize this primitive to obtain a multivalue byzantine agreement. In Section 4 we show how to implement BOOST with statistical security, which is shown mainly as a warmup. In Section 5 we show how to implement it with perfect security.

3.2 Main Protocol: Multivalue Byzantine Agreement

Protocol 3.1: Multivalue Byzantine Agreement

- **Input:** Each party P_i holds $f_i(x)$ of degree d as input.
- **The protocol:**
  1. **BOOST → Dissemination → Reliable Agreement:**
     a. Run BOOST protocol with input $f_i(x)$.
     b. **Upon** P_i sets $\mathsf{output}_i \neq \perp$ in BOOST, enter asynchronous data dissemination (Protocol 2.2). If $\mathsf{output}_i = f_i^{(1)}(x)$, then enter with that input; If $\mathsf{output}_i = \mathsf{proceed}$ then enter with $\perp$.
     c. **Upon** dissemination terminates with output $f_i^{(2)}(x)$, run reliable agreement (Protocol 2.3) with that input $f_i^{(2)}(x)$.
  2. **Asynchronous Binary Byzantine Agreement:** The parties run a single instance of Byzantine Agreement, where the input of each party is determined once, according to which event occurs first:
     a. **Upon** reliable agreement terminates with some output, enter BA with input 1.
     b. **Upon** BOOST activates $\mathsf{detect}_i = 1$, enter BA with input 0.
- **Output:**
  1. **Upon** BA terminates with output 0, terminate and output $\perp$.
  2. **Upon** BA terminates with output 1, output the output of reliable agreement and halt.

The following theorem is proven in the appendix:

▶ **Theorem 3.2.** *Protocol 3.1 is a multivalued byzantine agreement protocol, tolerating t corrupted parties.*

4 BOOST with Statistical Security

We refer to Section 1.2 for an overview of the following protocol.

Protocol 4.1: BOOST with Statistical Security

- **Input:** Each party enters with an input $f_i(x)$ – a polynomial of degree at most t.
- **Initialization:** Initialize $\mathsf{output}_i = \perp$, $g_i(x) = \perp$, $\mathsf{detect}_i = \perp$. Moreover, initialize an array C_i of size n, and a list T_i.

The protocol:

1. **Exchange:**
   a. Each party P_i chooses $r_i \leftarrow \mathbb{F}$ uniformly at random, and send to every P_j the message $(\texttt{challenge}, r_i)$.
   b. **Upon** receiving $(\texttt{challenge}, r_j)$ from P_j, set $C_i[j] = r_j$. Moreover, reply P_j with the message $(\texttt{Exchange}, f_i(r_j))$.
2. **Getting $2t+1$ support or send detect:**
   a. **Upon** receiving $(\texttt{Exchange}, u_j)$ from P_j, add (j, u_j) to the list T_i. If $u_j \neq f_i(r_i)$, then add j to DA_i.
   b. **Upon** $|\mathsf{DA}_i| \geq t+1$, then send to all parties the message detect.
   c. **Upon** T_i containing the same evaluation u_i from at least $t+1$ parties, send $(\texttt{support}, r_i, u_i)$ to all the parties.
   d. **Upon** receiving $2t+1$ messages $(\texttt{support}, r_j, u_j)$ such that $C_i[j] = r_j$ and $f_i(C_i[j]) = u_j$, then set $g_i(x) = f_i(x)$. Send to each P_k the message $(\mathsf{YourPoint}, g_i(C_i[k]))$.
   e. **Upon** receiving $t+1$ messages $(\texttt{support}, r_j, v_j)$ such that $C_i[j] \neq r_j$ or $v_j \neq f_i(r_j)$, then send detect to all parties.
3. **Reconstruction:**
   a. **Upon** receiving $t+1$ messages $(\mathsf{YourPoint}, u)$ with the same u, and P_i did not previously send $(\mathsf{MyPoint}, \cdot, \cdot)$ message, then send $(\mathsf{MyPoint}, r_i, u)$ to all. If $u \neq f_i(r_i)$, then send detect message to all (if not sent yet).
   b. **Upon** receiving a message $(\mathsf{MyPoint}, r_j, u_j)$ then add (r_j, u_j) to S_i. If $f_i(r_j) \neq u_j$, then add j to DA_i. (recall that **upon** $|\mathsf{DA}_i| \geq t+1$ then P_i sends detect message to all.)
   c. **Upon** $\mathsf{RSDec}(S_i, t) = f_i'(x)$, and $f_i'(r_j) = u_j$ for $2t+1$ points in S_i, then set $g_i(x) = f_i(x)$; Send $\texttt{HaveOutput}$ to all.
4. **Have output and termination:**
   a. **Upon** receiving $\texttt{HaveOutput}$ messages from $2t+1$ distinct parties, send $\texttt{done}$ to all.
   b. **Upon** receiving $\texttt{done}$ messages from $t+1$ distinct parties, send $\texttt{done}$ message to all.
5. **Detection:**
   a. **Upon** receiving detect messages from $t+1$ distinct parties, and if not sent detect message yet, send detect to all.
   b. **Upon** receiving detect messages from $2t+1$ distinct parties, set the output tape $\mathsf{detect}_i = 1$.
6. **Setting output:**
   a. **Upon** receiving $\texttt{done}$ messages from $2t+1$ distinct parties, then: If $g_i(x) \neq \perp$ then set $\mathsf{output}_i = g_i(x)$. Otherwise set $\mathsf{output}_i = \mathsf{proceed}$.

The following theorem is proven in the appendix:

▶ **Theorem 4.1.** *Protocol 4.1 is a* BOOST *protocol (as per Definition 3.1) where validity and set output hold unconditionally (with probability 1), and the Detect or correct and detect properties hold with probability* $1 - \frac{n^3}{|\mathbb{F}|}$.

5 BOOST with Perfect Security

In this section, we show how to implement the BOOST protocol for the perfect setting. We already provided an overview for this protocol in Section 1.2, and so we provide the formal description next.

5.1 The Entire Protocol

Protocol 5.1: BOOST **with Perfect Security**

- **Input:** Each party P_i starts with $f_i(x)$ of degree at most $t/7$ over $\mathbb{F}$.
- **Initialization:** The following variables are shared between the different sub-protocols. Initialize $\mathsf{Others}_i, \mathsf{My}_i, \mathsf{DA}_i = \emptyset$. Initialize $\mathsf{S}_i^1, \mathsf{S}_i^2, \mathsf{S}_i^3, \mathsf{LD}_i = \emptyset$.
- **The protocol:** Run all sub-protocols in parallel (Sub-protocol 5.2, 5.3, 5.4, 5.5, 5.7, 5.6), while using the same internal variables.
- **Output:** The protocol does not terminate, but each party P_i might eventually write to the output tapes output_i or detect_i (or both).

Sub-Protocol 5.2: Exchange Phase

1. Send to each P_j the message $(\texttt{Exchange}, f_i(i), f_i(j))$.
2. **Upon** receiving $(\texttt{Exchange}, u_j, v_j)$ from P_j:
   a. Add the points (j, u_j) to Others_i and (j, v_j) to My_i.
   b. If $|\mathsf{Others}_i| \geq 2t + 1$, run $L_i = \mathsf{ListDecode}(\mathsf{Others}_i, t/7, t + 1)$, and set $\mathsf{S}_i^1 = \mathsf{S}_i^1 \cup L_i$.
   c. If $u_j \neq f_i(j)$ or $v_j \neq f_i(i)$ then add j to DA_i.
3. **Upon** $|\mathsf{DA}_i| \geq t + 1$ send detect to all.
4. **Upon** receiving detect message from P_j, add j to DA_i.

Sub-Protocol 5.3: Filter Phase

1. **Upon** adding a polynomial $g_i(x)$ to S_i^1:
   a. Let $P[g_i(i)] = \{j \in [n] \mid (j, v) \in \mathsf{My}_i\}$.
   b. **Upon** $|P[g_i(i)]| \geq t + 1$, then send $(\texttt{MyPotentialPoint}, g_i(i))$ to all parties. If $g_i(i) \neq f_i(i)$, then send detect to all. Moreover, note that each party sends at most two $\texttt{MyPotentialPoint}$ messages.
2. **Upon** receiving $(\texttt{MyPotentialPoint}, u_j)$ from P_j, add (j, u_j) to T_i.
3. **Upon** (a) $g_i(x) \in \mathsf{S}_i^1$; (b) $g_i(x)$ agrees with $2t + 1$ points in T_i, then add $g_i(x)$ to S_i^2.
4. **Upon** a polynomial $g_i(x)$ added to S_i^2, but $g_i(x) \neq f_i(x)$, then send detect to all.

Sub-Protocol 5.4: Detect Simple Conflicts

1. **Upon** adding a polynomial $g_i(x)$ to S_i^2:
   a. Send to each party P_j the message $(\texttt{YourPoint}, g_i(j))$.
   b. If $g_i(j) \neq f_i(j)$, then send detect message to all.
2. **Upon** receiving $(\texttt{YourPoint}, u_j)$ from P_j, add (j, u_j) to S_i.
3. **Upon** (a) $g_i(x) \in \mathsf{S}_i^2$; (b) S_i containing $(j, g_i(i))$ from at least $2t + 1$ distinct parties, then add $g_i(x)$ to S_i^3.

Sub-Protocol 5.5: Resolve Conflicts

1. **Upon** adding a polynomial $h_i(x)$ to S_i^3:
   a. Send to each party P_j the message $(\texttt{MyPoint}, h_i(j))$.
2. **Upon** $|\mathsf{S}_i^3| > 1$, send detect to all.
3. **Upon** receiving first $(\texttt{MyPoint}, u_j)$ from P_j, add (j, u_j) to R_i. (If a party sends more than one point, add to R_i only the first point it sent.)
4. **Upon** R_i containing messages from $2t+1$ distinct parties:
   a. Let $\mathsf{LD}_i := \mathsf{ListDecode}(R_i, t/7, t+1)$.
   b. If $|\mathsf{S}^3 \cup \mathsf{LD}_i| > 1$ then
      i. Find $i' = i + c \cdot n$ for the first integer c for which $|\{h(i') \mid h \in \mathsf{S}_i^3 \cup \mathsf{LD}_i\}| = |\mathsf{S}_i^3 \cup \mathsf{LD}_i|$, i.e., $h(i')$ is different for every polynomial in $\mathsf{S}_i^3 \cup \mathsf{LD}_i$.
      ii. Send to all parties $(\texttt{challenge}, i')$.
      iii. **Upon**
         A. receiving $2t+1$ responses $(\texttt{response}, v)$ from distinct parties, that all agree on v;
         B. $\left|\{h_i(x) \in \mathsf{S}_i^3 \mid h_i(i') = v\}\right| = 1$; and
         C. $g_i(x) = \perp$:
         Set $g_i(x)$ to be that unique polynomial $h_i(x)$.
   c. **Upon** seeing $(\texttt{challenge}, j')$ from j: if there exists $g_i(x) \in \mathsf{S}_i^3$ and you did not yet respond with $(\texttt{response}, g_i(j'))$ then send P_j $(\texttt{response}, g_i(j'))$.
   d. **Upon** (a) $|\mathsf{S}^3 \cup \mathsf{LD}_i| = 1$; (b) $\mathsf{S}_i^3 = \{h_i(x)\}$; (c) $g_i(x) = \perp$: set $g_i(x) = h_i(x)$.

Sub-Protocol 5.6: Totality and Abundance Phase

1. **Upon** $g_i(x) \neq \perp$, set `HaveOutput` to all.
2. **Upon** receiving `HaveOutput` from distinct $2t+1$ parties, send `done` to all.
3. **Upon** receiving `done` messages from distinct $t+1$ parties, and did not send `done` message yet, send `done` to all.
4. **Upon** receiving `done` messages from distinct $2t+1$ parties, set $\mathsf{output}_i = g_i(x)$ if $g_i(x) \neq \perp$ and set $\mathsf{output}_i = \mathsf{proceed}$ otherwise.

Sub-Protocol 5.7: Detection

1. **Upon** receiving $t+1$ detect message, send detect to all.
2. **Upon** receiving detect messages from $2t+1$ distinct parties, set $\mathsf{detect}_i = 1$.

In Section 5.2 we prove the following theorem:

▶ **Theorem 5.1.** *Protocol 5.1 is a* BOOST *protocol (as per Definition 3.1), tolerating at most t corrupted parties. The protocol is perfectly secure and requires $O(n^2 \log n)$ communication.*

The theorem is proven by explicitly specifying the properties each sub-protocol achieves. We provide a formal proof for those sub-protocols in the full version [2].

▷ Claim 5.2. Sub-protocol 5.2 (Exchange phase) satisfies the following properties:

1. **Validity:** If all honest start with the same input $F(x)$, then no honest send detect message and eventually all honest add $F(x)$ to S_i^1.
2. **Locally small:** Each honest party P_i adds no more than 3 polynomials to S_i^1.
3. **Boost:** If there exists a set of at least $t+1$ honest parties that start with the same polynomial $F(x)$, then each honest party eventually adds $F(x)$ to its S_i^1.
4. **Detect:** If there is no set of $t+1$ honest parties with the same polynomial, then all honest parties eventually detect.

▷ Claim 5.3. Sub-Protocol 5.3 (Filter phase) satisfies the following properties:
1. **Validity:** if all honest parties have $F(x)$ in their input, then no honest party detects, and each honest party eventually adds $F(x)$ to S_i^2. Moreover, no other polynomial is ever added to S_i^2.
2. **Detect or correct**: If there exists a set of $t+1$ honest parties that start with the same input $F(x)$, then each honest party eventually has $F(x) \in \mathsf{S}_i^2$. Moreover, honest parties that do not hold $F(x)$ as input will detect.

Now the problem is that we may get two polynomials in S_i^2. The above protocol guarantees that any $\neq F(x)$ input party will detect, but they are the minority so we may just get just $|B| < t+1$ detects from this.

The next step makes sure that if all honest eventually have $F(x) \in \mathsf{S}_i^2$ and some honest adds $G(x) \neq F(x)$ to S_i^3 with $G(i) \neq F(i)$ then at least $t+1-|B|$ additional parties that started with $F(x)$ detect. For a total desired $t+1$ honest detections.

▷ Claim 5.4. Sub-protocol 5.4 (Detect Simple Conflicts) satisfies the following properties:
1. **Validity:** If all honest eventually have $\mathsf{S}_i^2 = \{F(x)\}$ then no honest detects and eventually all honest have $\mathsf{S}_i^3 = \{F(x)\}$.
2. **Detect or correct:** Assume there exists a set of $t+1$ honest parties that hold the same $F(x)$ as input. Then; (a) all honest parties eventually add $F(x)$ to S_i^3, (b) if some honest party i adds $G(x) \neq F(x)$ to S_i^3 with $G(i) \neq F(i)$, then all honest parties eventually detect.

So now the remaining problem is if all honest that add $G(x) \neq F(x)$ to S_i^3 have $G(i) = F(i)$.

The next protocol handles this case. First, it makes sure that the parties with $G(i) = F(i)$ will see $F(x)$ in in a new List Decoding, then it resolves this by having them choose a new index where $G(i') \neq F(i')$.

▷ Claim 5.5. Sub-protocol 5.5 (Resolve conflicts) satisfies the following properties:
1. **Validity:** If all honest parties eventually have $\mathsf{S}_i^3 = \{F(x)\}$, then all eventually set $g_i(x) = F(x)$.
2. **Detect or correct:** If all honest eventually have $F(x) \in \mathsf{S}_i^3$, all honest will eventually have set $g_i(x) = F(x)$, or $t+1$ honest parties eventually detect.

▷ Claim 5.6. Sub-Protocol 5.6 (Totality and Abundance Phase) The protocol satisfies the following properties:
1. **Totality:** If an honest set output then all will eventually set output.
2. **Abundance:** If an honest set output then at least $t+1$ must have $g_i(x) \neq \perp$.
3. **Validity:** If all honest set $g_i(x) \neq \perp$, then all will set output output_i.

▷ Claim 5.7. Sub-protocol 5.7 (Detection phase) satisfies the following conditions:
1. If parties set $\mathsf{detect}_i = 1$, then at least one honest party initiated a detect message throughout the protocol.
2. If $t+1$ honest parties initiate detect message, then all honest parties eventually set $\mathsf{detect}_i = 1$.

5.2 Putting it All Together: Proving Theorem 5.1

▶ **Theorem 5.8** (Theorem 5.1, restated)**.** *Protocol 5.1 is a* BOOST *protocol (as per Definition 3.1), tolerating at most t corrupted parties. The protocol is perfectly secure and requires $O(n^2 \log n)$ communication.*

Proof. We show that each of the properties must hold.

Validity. We show that if all honest parties start with the same input $F(x)$, then honest parties do not detect, and at least $t+1$ honest parties output $F(x)$ and the rest output detect.

1. From the validity condition of Claim 5.2, no honest party detects in that subprotocol, and eventually all honest parties add $F(x)$ to S_i^1.
2. From the validity condition of Claim 5.3, no honest party detects, and each party eventually have $\mathsf{S}_i^2 = \{F(x)\}$.
3. From the validity condition of Claim 5.4, all honest parties eventually have $\mathsf{S}_i^3 = \{F(x)\}$ and do not trigger detect.
4. From the validity condition of Claim 5.5, all honest parties eventually set $g_i(x) = F(x)$.
5. From the validity condition of Claim 5.6, when all honest parties set $g_i(x) \neq \perp$, then all set output output_i.
6. If one honest party sets output output_i, then from the Abundance condition of Claim 5.6, at least $t+1$ honest parties must have $g_i(x) \neq \perp$. The only value that this could be is $F(x)$. The rest (which might not have $g_i(x) \neq \perp$) set their output to proceed.

Set output. If an honest party sets its output, then all honest parties eventually set output – this follows directly from the abundance condition in Claim 5.6.

Detect. If there is no set of $t+1$ honest parties with the same input, then all honest parties eventually detect. This follows directly from the detect condition in Claim 5.2.

Detect or correct. We show that if at least $t+1$ honest parties hold the same polynomial $F(x)$ as input, then (a) either eventually all honest parties detect; or (b) $t+1$ honest parties set their output to $F(x)$ and the remaining set output to proceed.

We show that if there is no detect, then $t+1$ honest parties set their output to $F(x)$, and the remaining set to proceed:

1. From the boost condition in Claim 5.2, all honest parties eventually add $F(x)$ to S_i^1.
2. From the detect or correct condition of Claim 5.3, each honest party eventually has $F(x) \in \mathsf{S}_i^2$. Moreover, honest parties that do not hold $F(x)$ as input trigger detect. However, we assume that those are less than t parties.
3. From the detect or correct condition of Claim 5.4, then all honest parties eventually add $F(x)$ to S_i^3; moreover, if some honest party adds $G(x) \neq F(x)$ to S_i^3 with $G(i) \neq F(i)$, then all honest parties detect. Therefore, since we assume that there is no detect, the only polynomials $G(x)$ that could be added to S_i^3 must satisfy $G(i) = F(i)$.
4. From the detect or liveness property of Claim 5.5, if all honest parties eventually have $F(x) \in \mathsf{S}_i^3$, then all honest parties eventually set $g_i(x)$. Moreover, from detect or correct, if some party sets $g_i(x)$ to be $G(x) \neq F(x)$, then all honest parties must detect. Assuming no detect, we must conclude that all honest parties set $g_i(x) = F(x)$.
5. From the validity condition of Claim 5.6, when all honest parties set $g_i(x) \neq \perp$, then all set output output_i.
6. If one honest party sets output output_i, then from the Abundance condition of Claim 5.6, at least $t+1$ honest parties must have $g_i(x) \neq \perp$. We showed that if this value is not $F(x)$, then we must eventually detect. Moreover, the rest (which might not have $g_i(x) \neq \perp$ yet) set their output to proceed.

Efficiency. From Claim 5.2, each party has at most 3 polynomials in S_i^1. This bounds all the potential points that a party might send, and by inspection, each party sends at most $O(n)$ points on $\mathbb{F}$ to every other party. We conclude that all honest parties send/receive $O(n^2 \log n)$ bits. In case of $L \in \Omega(n \log n)$, the protocol requires encoding the input as polynomials, and requires $O(nL + n^2 \log n)$ communication. ◀

References

1 Ittai Abraham and Gilad Asharov. Gradecast in synchrony and reliable broadcast in asynchrony with optimal resilience, efficiency, and unconditional security. In *PODC '22: ACM Symposium on Principles of Distributed Computing, 2022*, pages 392–398. ACM, 2022. doi:10.1145/3519270.3538451.

2 Ittai Abraham and Gilad Asharov. ABEL: Perfect asynchronous byzantine extension from list-decoding. Cryptology ePrint Archive, Paper 2025/1488, 2025. URL: https://eprint.iacr.org/2025/1488.

3 Ittai Abraham, Gilad Asharov, and Anirudh Chandramouli. Simple is COOL: graded dispersal and its applications for byzantine fault tolerance. In Raghu Meka, editor, *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*, volume 325 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ITCS.2025.1.

4 Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Asymptotically free broadcast in constant expected time via packed VSS. In *Theory of Cryptography - 20th International Conference, TCC 2022*, volume 13747 of *Lecture Notes in Computer Science*, pages 384–414. Springer, 2022. doi:10.1007/978-3-031-22318-1_14.

5 Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, pages 381–391, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538426.

6 Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. *Distributed Comput.*, 36(1):3–28, 2023. doi:10.1007/s00446-022-00428-8.

7 Jinyuan Chen. Optimal error-free multi-valued byzantine agreement. In *DISC 2021*, volume 209 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.17.

8 Jinyuan Chen. Ociormvba: Near-optimal error-free asynchronous mvba, 2024. doi:10.48550/arXiv.2501.00214.

9 B. A. Coan and R. Turpin. Extending binary byzantine agreement to multivalued byzantine agreement. Technical report, Massachusetts Institute of Technology, USA, 1984.

10 Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2721. ACM, 2021. doi:10.1145/3460120.3484808.

11 Sourav Das, Zhuolun Xiang, and Ling Ren. Balanced quadratic reliable broadcast and improved asynchronous verifiable information dispersal. *IACR Cryptol. ePrint Arch.*, page 52, 2022. URL: https://eprint.iacr.org/2022/052.

12 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 1982*, pages 132–140. ACM, 1982. doi:10.1145/800220.806690.

13 Mose Mizrahi Erbes and Roger Wattenhofer. Extending asynchronous byzantine agreement with crusader agreement, 2025. doi:10.48550/arXiv.2502.02320.

14 Hanwen Feng, Zhenliang Lu, Tiancheng Mai, and Qiang Tang. Faster hash-based multi-valued validated asynchronous byzantine agreement. Cryptology ePrint Archive, Paper 2024/479, 2024. to apper in DSN 25. URL: https://eprint.iacr.org/2024/479.

15 Matthias Fitzi and Martin Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 163–168, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1146381.1146407.

16 Venkatesan Guruswami. Algorithmic results in list decoding. *Found. Trends Theor. Comput. Sci.*, 2(2):107–195, January 2007. doi:10.1561/0400000007.

17 Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, PODC '20, pages 129–138, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3382734.3405707.

18 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t < n/3, o(n2) messages, and o(1) expected time. *J. ACM*, 62(4), September 2015. doi:10.1145/2785953.

19 Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved Extension Protocols for Byzantine Broadcast and Agreement. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2020.28.

20 Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *J. Complex.*, 13(1):180–193, March 1997. doi:10.1006/jcom.1997.0439.

A Omitted Proofs for Section 2

▶ **Theorem A.1** (Theorem 2.6, restated). *Protocol 2.3 is a reliable agreement protocol tolerating $t < n/3$ corrupted parties. It requires $O(nL + n^2 \log N)$ communication for an input of size L.*

Proof. We show that each one of the properties hold:

- **Termination:** A party terminates only after the dissemination terminates. As guaranteed by dissemination, if one party terminates, then all honest parties eventually terminate.
- **Validity:** If all honest parties enter the protocol with the same input $F(x)$, then dispersal is guaranteed to terminate, and at least $t + 1$ honest parties terminate with the same output $F(x)$, and the rest terminate with $\perp$. The validity of dissemination then guarantees that all honest parties terminate and output $F(x)$.
- **Agreement:** Assume an honest party terminates the protocol with output $F'(x)$. This implies that dispersal must have terminated. Dispersal guarantees weak agreement – if an honest party terminates, then at least $t + 1$ honest parties terminates with the same polynomial $F''(x)$, and the rest might terminate with either $F''(x)$ or $\perp$. The parties then run dissemination, which then guarantees that all honest parties terminate with the output $F''(x)$. It thus hold that $F''(x) = F'(x)$, and that all honest parties must terminate with $F'(x)$. ◀

B Omitted Proofs for Section 3

▶ **Theorem B.1** (Theorem 3.2, restated). *Protocol 3.1 is a multivalued byzantine agreement protocol, tolerating t corrupted parties.*

Proof. We show that each one of the properties hold.

Validity. If all honest parties start with the same input $F(x)$, then:

- The validity property of BOOST (Definition 3.1) guarantees that no honest party triggers $\mathsf{detect}_i = 1$; moreover, all honest parties must eventually set $\mathsf{output}_i = F(x)$.

- The validity property of data dissemination then guarantees that all parties eventually set $f_i^{(2)}(x) = F(x)$.
- The validity reliable agreement guarantees that all honest parties must receive $F(x)$ as output.
- We get that no honest party enters 0 to the Byznatine Agreement, as no honest party ever set $\mathsf{BOOST.detect}_i = 1$. All honest parties start the BA with the same input 1. The validity property of BA then guarantees that all honest parties must output 1.
- All honest parties output the output of the second dissemination, i.e., $F(x)$.

Termination. To show termination, we show that the parties must eventually enter the BA. To show that the parties must eventually enter the BA, we have three cases to consider:

1. If all honest parties hold the same input $F(x)$, then as guaranteed from the validity property, all terminate, and in particular also enter the BA.
2. If there is no set of $t+1$ honest parties with the same input $F(x)$, then the detect property of BOOST guarantees that all honest parties eventually activate detect. Therefore, all honest parties must eventually enter the BA (some might enter earlier with different input).
3. If there is a set of $t+1$ honest parties with the same input $F(x)$, then the detect or correct property of BOOST guarantees that either:
   a. **Detect:** Eventually, all honest parties activate their detect output tape. In that case, parties have input to the BA.
   b. **Correct:** At least $t+1$ honest parties set their output tape to $F(x)$ and the remaining to proceed. The validity of dissemination (Definition 2.3) property then guarantees that all honest parties must terminate with output $F(x)$. The protocol then proceeds to reliable agreement, which now its validity guarantees that all output $F(x)$. We conclude that all honest parties eventually can enter the BA with 1.

Agreement. We show that all honest parties must have the same output. In particular, this follows from BA: all honest parties must receive the exact same output from BA. There are two cases:

- If the output is 0, then all honest parties terminate and output $\perp$.
- If the output is 1, then there exists an honest party that inputs 1 to the BA. A party inputs 1 to the BA only if its reliable agreement terminated. If an honest party terminates the reliable agreement, then all honest parties would eventually terminate the reliable agreement with some output $F'(x)$, then eventually all honest parties terminate with output $F'(x)$. ◀

C Omitted Proofs for Section 4

▶ **Theorem C.1** (Theorem 4.1, restated). *Protocol 4.1 is a* BOOST *protocol (as per Definition 3.1) where validity and set output hold unconditionally (with probability 1), and the Detect or correct and detect properties hold with probability* $1 - \frac{n^3}{|\mathbb{F}|}$.

Proof. We show that the protocol satisfies all those properties.

Validity. If all honest parties enter with the same polynomial $F(x)$, then no matter what challenge each party chooses, the parties will agree with each other. Each party P_i might add to DA_i only corrupted parties, and thus $|\mathsf{DA}_i| \leq t$. Each party never sends `support` message on a value that is not on F, and each party sees $2t+1$ support messages to points on F. Therefore, each party P_i sends $(\mathsf{YourPoint}, F(C_i[k]))$ to each P_k. Each P_k receives

$2t+1$ YourPoint messages with the same value, and therefore sends $(\mathsf{MyPoint}, r_k, F(r_k))$ to all. Each party will eventually see $2t+1$ points on $F(x)$, and the unique decoding is exactly $F(x)$. Therefore, each party eventually sets $g_i(x) = F(x)$, and send `HaveOutput` message.

However, parties might set output before setting $g_i(x) = F(x)$. A party sets its output when receiving $2t+1$ `done` messages. This implies that $t+1$ honest parties sent a `done` message. A party sends a `done` message if it received $t+1$ `done` messages, or if it received $2t+1$ `HaveOutput` messages. In the latter case, $t+1$ sent `HaveOutput` messages; In the former case, this implies that at least one honest party initiated a `done` message – which again implies that there are $t+1$ honest parties that sent `HaveOutput` message. We showed that all honest parties eventually set $g_i(x) = F(x)$, send `HaveOutput` message, and therefore when a party sets its output, there are at least $t+1$ honest parties that already set $g_i(x) = F(x)$, and the rest would set their output to proceed.

We also claim that no honest party send detect message. An honest party P_j might send such a message in the following cases:

1. $|\mathsf{DA}_j| \geq t+1$. However, since all honest parties agree with their exchange messages, a party might add to DA_j only corrupted parties, and therefore $|\mathsf{DA}_j| \leq t$.
2. A party sends a detect message if it receives $t+1$ messages $(\texttt{support}, r_j, v_j)$ such that $C_i[j] \neq r_j$ or $v_j \neq f_i(r_j)$. However, an honest party P_i sends `support` only after seeing at least $t+1$ `Exchange` messages with the same (r_i, u_i). Since all honest parties have the same input, the only value that can reach $t+1$ cardinality is a value on F. Therefore, the only $(\texttt{support}, r_i, v_i)$ for which $C_j[i] \neq r_i$, or $v_i \neq f_j(r_i)$ are messages from corrupted parties, and therefore it can receive at most t of those.
3. An honest party might send detect upon receiving $t+1$ messages $(\mathsf{YourPoint}, u)$ with the same u, but $u \neq f_j(r_j)$. As previously, honest parties send YourPoint messages only on values on F, and therefore no other value can reach cardinality $t+1$.
4. Once again, parties add i to DA_j if they receive $(\mathsf{MyPoint}, r_i, u_i)$ for which $f_j(r_i) \neq u_i$. However, since all honest parties send MyPoint only on values on F, DA_j never reaches cardinality $t+1$.

We conclude that no honest party ever sends detect message.

Set output. An honest party sets its output only after receiving `done` messages from $2t+1$ distinct parties. This implies that $t+1$ honest parties must send `done` messages. Those parties send that message to all parties, and therefore each honest party must eventually receive $t+1$ `done` messages, and thus also send `done` messages. As a result, each honest party must eventually receive `done` messages from $2t+1$ distinct parties, and set its output.

Bad event. Before proceeding to show the detect or correct property that and the detect property, we first define a bad event. let $\mathsf{Bad}_{i,j}$ denote the event in which P_i and P_j hold input $f_i(x) \neq f_j(x)$, respectively, but P_i chooses r_i such that $f_i(r_i) = f_j(r_i)$. In that case, P_i might not recognize that P_j does not hold the same polynomial. We can bound $\Pr[\mathsf{Bad}_{i,j} = 1] \leq \frac{d}{|\mathbb{F}|}$. This is because the two polynomials agree on d points, and P_i chooses its challenge uniformly at random. Moreover, let Bad denote the event in which there exists some i, j for which $\mathsf{Bad}_{i,j}$ occurs. We have that

$$\Pr[\mathsf{Bad} = 1] \leq \sum_{i,j} \Pr[\mathsf{Bad}_{i,j} = 1] \leq \frac{n^2 d}{|\mathbb{F}|} \leq \frac{n^3}{|\mathbb{F}|} .$$

We note that here we assume that the adversary must choose the input of all honest parties before the protocol starts. To handle adaptive inputs, a variant with additional rounds are needed (see [1]). Since our goal with statistical security is as a warm-up to the perfect security case, we defer this variation to a full version. We proceed with the analysis assuming that Bad does not occur.

Detect or correct. We assume that $t+1$ honest parties start with the same input $F(x)$. We show that, unless there is a detection, at least $t+1$ honest parties set their output to $F(x)$, and the rest set their output to proceed.

There are two cases to consider:

1. Suppose some honest party P_i receives $2t+1$ messages $(\texttt{support}, r_i, u_i)$ such that P_i previously sent $(\texttt{challenge}, r_i)$ and $u_i \neq F(r_i)$. Then, we show that all honest parties eventually detect.

   Specifically, if an honest party sees $2t+1$ such messages, then $t+1$ honest parties sent $(\texttt{support}, r_i, v_i)$ to all. All honest parties would see those messages, specifically also the parties that hold $F(x)$. Due to Step 2e, they would see $t+1$ values v_i for which $v_i \neq F(r_j)$, and thus would send detect to all. Since $t+1$ parties hold $F(x)$, $t+1$ honest parties send detect, which leads to all parties detect.
2. No honest parties gets $2t+1$ support on a value $F'(r_i) \neq F(r_i)$. Clearly, all honest parties must receive $2t+1$ messages $(\texttt{support}, r_i, F(r_i))$: Each honest party sends $(\texttt{challenge}, r_i)$ to all parties; it would receive from the parties in F the same value $(\texttt{Exchange}, F(r_i))$, and thus send $(\texttt{support}, r_i, F(r_i))$ to all. The $t+1$ honest parties holding F would eventually receive $2t+1$ messages $(\texttt{support}, r_i, F(i))$ (with difference indices i), and therefore all would send to each P_i $(\mathsf{YourPoint}, F(r_i))$. According to our assumption, no other value receives $2t+1$, therefore, all messages honest parties send are on F. Each party would receive $t+1$ $(\mathsf{YourPoint}, F(r_i))$ with the same value, and therefore would send $(\mathsf{MyPoint}, r_i, F(r_i))$ to all. The only polynomial that can be decoded is $F(x)$.

Detect. We show that if there is no set of $t+1$ honest parties that hold the same polynomial, and assuming that Bad does not occur, then all honest parties set $\mathsf{detect}_i = 1$. Since there is no common input for $t+1$ honest parties, each honest party must hold a different input than at least $t+1$ parties. Assuming that Bad does not occur, each party P_i adds at least $t+1$ indices to its DA_i set, and eventually send detect message. All honest parties eventually send detect, and activate $\mathsf{detect}_i = 1$. ◀

D Omitted Proofs for Section 5

For the proofs of section 5, please refer to the full version of this paper [2].

On the Efficiency of Dynamic Transaction Scheduling in Blockchain Sharding

Ramesh Adhikari
School of Computer and Cyber Sciences, Augusta University, GA, USA

Costas Busch
School of Computer and Cyber Sciences, Augusta University, GA, USA

Miroslav Popovic
Faculty of Technical Sciences, University of Novi Sad, Serbia

Abstract

Sharding is a technique to speed up transaction processing in blockchains, where the n processing nodes in the blockchain are divided into s disjoint groups (shards) that can process transactions in parallel. We study dynamic scheduling problems on a shard graph G_s where transactions arrive online over time and are not known in advance. Each transaction may access at most k shards, and we denote by d the worst distance between a transaction and its accessing (destination) shards (the parameter d is unknown to the shards). To handle different values of d, we assume a locality sensitive decomposition of G_s into clusters of shards, where every cluster has a *leader shard* that schedules transactions for the cluster. We first examine the simpler case of the *stateless model*, where leaders are not aware of the current state of the transaction accounts, and we prove a $O(d \log^2 s \cdot \min\{k, \sqrt{s}\})$ competitive ratio for latency. We then consider the *stateful model*, where leader shards gather the current state of accounts, and we prove a $O(\log s \cdot \min\{k, \sqrt{s}\} + \log^2 s)$ competitive ratio for latency. Each leader calculates the schedule in polynomial time for each transaction that it processes. We show that for any $\epsilon > 0$, approximating the optimal schedule within a $(\min\{k, \sqrt{s}\})^{1-\epsilon}$ factor is NP-hard. Hence, our bound for the stateful model is within a poly-log factor from the best possibly achievable. To the best of our knowledge, this is the first work to establish provably efficient dynamic scheduling algorithms for blockchain sharding systems.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Blockchain, Blockchain Sharding, Dynamic Transaction Scheduling

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.2

Related Version *Extended Version*: `https://arxiv.org/abs/2508.07472` [4]

Funding This paper is supported by NSF grant CNS-2131538.

1 Introduction

Blockchains are known for their special features, such as fault tolerance, transparency, non-repudiation, immutability, and security, and have been used in various applications and domains [16]. However, a drawback of blockchains is that the size of the blockchain network may impact the latency and throughput of transaction processing. To append a new block in a blockchain network, the participating nodes reach consensus, which is a time and energy-consuming process [2]. Moreover, each node is required to process and store all transactions, which leads to scalability issues in the blockchain system. *Sharding* protocols have been proposed to address the scalability and performance issues of blockchains [15, 21, 11, 1], which divide the overall blockchain network into smaller groups of nodes called *shards* that allow for processing transactions in parallel. In the sharded blockchain, independent transactions are processed and committed in multiple shards concurrently, which improves the blockchain


© Ramesh Adhikari, Costas Busch, and Miroslav Popovic;
licensed under Creative Commons License CC-BY 4.0



system's throughput. However, most of the existing sharding protocols [15, 13, 21, 1] do not provide formal analysis for the scheduling time complexity (i.e. how fast the transactions can be processed).

We consider a blockchain system consisting of n nodes, which are further divided into s shards, where each shard consists of n/s nodes. Shards are connected in a graph network G_s with a diameter D, and each shard holds a subset of the objects (transaction accounts). We assume that transactions are distributed across the shards, and each transaction accesses at most k accounts. A transaction T_i initially is in one of the shards, which is called the *home shard* for T_i. For simplicity, we consider each shard has one transaction at a time, and when that transaction gets processed (either commit or abort), a new transaction will be generated at the home shard. Similar to other sharding systems [11, 1, 2], each transaction T_i is split into subtransactions, where each subtransaction accesses an account. A subtransaction of T_i is sent to the *destination shard* that holds the respective account. We assume that the maximum distance between the home shard of a transaction and the respective destination shards in G_s is at most $d \leq D$. (The parameter d is not known to the system.)

All home shards process transactions concurrently. A problem occurs when *conflicting* transactions try to access the same account simultaneously. In such a case, the conflict prohibits the transactions from being committed concurrently and forces them to serialize [2]. Our proposed *scheduling algorithms* coordinate the home shards and destination shards to process the transactions (and respective subtransactions) in a conflict-free manner in polynomial time. Each destination shard maintains a local blockchain of the subtransactions that are sent to it. The global blockchain can be constructed (if needed) by combining the local blockchains at the shards [1].

We consider online dynamic transaction scheduling problem instances where transactions are not known a priori. Moreover, transactions may arrive online and continuously over time, which are generated by electronic devices or some crypto app that resides on shards. Our proposed schedulers determine the time step for each transaction $T_i \in \mathcal{T}$ to process and commit. The execution of our scheduling algorithm is partially synchronous, where communication delay is upper bounded by a system parameter. The goal of a scheduling algorithm is to efficiently process all transactions while minimizing the total execution time (makespan). Unlike previous sharding approaches [11, 13, 21], our scheduling algorithms are lock-free, namely, they do not require locking mechanisms for concurrency control.

We use the notion of *competitive ratio* [8] to determine the performance of our scheduling algorithms. The competitive ratio typically measures how well a given online algorithm performs compared to the best possible offline algorithm for a specific sequence of operations. However, in our model, the transactions generated in the future depend on the execution history. Hence, we define the competitive ratio to capture the volatile transaction history.

Contributions. To our knowledge, this is the first work to present provably efficient online transaction scheduling algorithms for blockchain sharding systems. We summarize our contributions as follows (also see Table 1):

- **Stateless Scheduling Model:** We first provide transaction scheduling algorithms for the stateless model, where a *leader shard* that is responsible for coordinating transaction execution, does not require knowledge of the current state of the accessed accounts. In this model, we provide two scheduling algorithms:
  - **Single-Leader Scheduler:** In this scheduling algorithm, one of the shards acts as the leader and all other shards send their transaction information to this leader, which determines the global transaction schedule. Our algorithm works in a partially

synchronous communication model, but for the sake of performance analysis purposes, we assume a synchronous model. Let the shard network be represented as a general graph G_s, where each transaction accesses at most k objects (shards). The maximum distance between home shards, accessed shards, and leader is denoted with d. Then, the single-leader scheduler achieves an $O(d \cdot \min\{k, \sqrt{s}\})$ competitive ratio with respect to the optimal scheduler. In the special case where G_s is a clique with unit distances (i.e., $d = 1$), the competitive ratio becomes $O(\min\{k, \sqrt{s}\})$.

  - **Multi-Leader Scheduler:** A drawback of the single-leader case is that the distance d involves also the position of the leader. On the other hand, in the multi-leader case, d only involves distances between home and respective destination shards. In this scheduler, multiple leaders process the transactions, which distribute the scheduling load among multiple shards. The multi-leader approach allows for a better adaptation to the value d without requiring knowledge of d and without involving distances to the leaders in the definition of d. This approach uses a hierarchical clustering technique [10] to cluster the shard network, which enables the independent scheduling and commitment of transactions within different clusters. This scheduler achieves a competitive ratio of $O(d \log^2 s \cdot \min\{k, \sqrt{s}\})$.

- **Stateful Scheduling Model:** We next consider a stateful model where the leader shard requires knowledge of the account states. Namely, a leader shard receives the transactions from the home shards (where transactions are initially generated), and then the leader shard first gathers the current state of the accounts from their corresponding account shards before scheduling and pre-committing the transactions. After receiving the state, the leader pre-commits the transactions locally and forwards the pre-committed batch to the destination shards. In this model, the single-leader scheduler achieves a competitive ratio of $O(\min\{k, \sqrt{s}\})$ and the multi-leader scheduler achieves a competitive ratio of $O(\log s \cdot \min\{k, \sqrt{s}\} + \log^2 s)$. Note that these competitive ratios do not depend on d (in contrast to the stateless model), which is the benefit of the stateful approach.
- **Approximation Hardness:** We also show that for any $\epsilon > 0$, obtaining competitive ratio $(\min\{k, \sqrt{s}\})^{1-\epsilon}$ is NP-hard. Hence, our bound for the stateful single-leader scheduler is asymptotically the best we can achieve in polynomial time, and the bound for the stateful multi-leader scheduler is within a poly-log factor of the best achievable.
- **Safety and Liveness Analysis:** We formally analyze the correctness of our proposed schedulers by proving both safety and liveness for the single-leader and multi-leader algorithms.

Paper Organization. The rest of this paper is structured as follows. Section 2 provides related works. Section 3 describes the preliminaries for this study and the sharding model. Section 4 presents a stateless scheduling model with single-leader and multi-leader scheduling algorithms. In Section 5, we provide the stateful single-leader and multi-leader scheduling algorithms. Finally, we give our conclusions in Section 6. Due to space limitations, some proofs are deferred to the appendix, while additional correctness analyses (safety and liveness) of our algorithms are provided in the extended version available on arxiv [4].

2 Related Work

To solve the scalability issue of blockchain, various sharding protocols [15, 21, 11, 17, 20, 6] have been proposed. These protocols have shown promising enhancements in the transaction throughput of blockchain by processing transactions in parallel in multiple shards. However,

Table 1 Comparison of our proposed online transaction scheduling algorithm's competitive ratio with related works [2, 3, 14]. The used notations are as follows: s represents a total number of shards, k denotes the maximum number of shards (objects) accessed by each transaction, d denotes the worst distance between any transaction (home shard) and its accessed objects (destination shard), b denotes the burstiness and c_1' represents some positive constant. (Note that the bounds in [2] are the actual transaction latencies).

	Proposed Results		**Related works**	
	Stateless Model	**Stateful Model**	**In [2]**	**In [3, 14]**
Problem	Dynamic Transaction	Dynamic Transaction	Dynamic Transaction	Batch Transaction
Focus	Performance	Perofrmance	Stability	Performance
Single Leader	$O\big(d \cdot \min\{k, \sqrt{s}\}\big)$	$O\big(\min\{k, \sqrt{s}\}\big)$	$36bd \cdot \min\{k, \lceil\sqrt{s}\rceil\}$	$O(kd)$
Multi-Leader	$O\big(d \log^2 s \cdot \min\{k, \sqrt{s}\}\big)$	$O\big(\log s \cdot \min\{k, \sqrt{s}\} + \log^2 s\big)$	$2 \cdot c_1' bd \log^2 s \cdot \min\{k, \lceil\sqrt{s}\rceil\}$	$O\big(kd \cdot \log d \log s\big)$
Com. Model	Partial-synchronous	Partial-synchronous	Synchronous	Synchronous

none of these protocols have specifically explored the theoretical analysis of online transaction scheduling problems in a sharding environment. To process transactions in parallel in the sharding model, some research work [13, 11] has used two-phase locking for concurrency control. However, locks are expensive because when one process locks shared data for reading/writing, all other processes attempting to access the same data set are blocked until the lock is released, which lowers system throughput. Moreover, locks, if not handled and released properly, may cause deadlocks. Our scheduling algorithms do not use locks, as concurrency control is managed by scheduling non-conflicting transactions in parallel. In [1] the authors propose lockless blockchain sharding using multi-version concurrency control. However, they lack a performance analysis, and they do not explore the benefits of locality and optimization techniques for transaction scheduling.

In a recent work [2] (see Table 1), the authors provide a stability analysis of blockchain sharding considering adversarial transaction generation. Their focus is on stability, not on performance, where they want to maintain a bounded pending transaction queue size and latency. They consider adversarial transaction generation, where at any time interval of duration t, the number of generated transactions using any object is bounded by $\rho t + b$, where $\rho \leq 1$ is the transaction injection rate per unit time and $b > 0$ is a burstiness injection parameter. They consider stateless scheduling model, and for the single leader scheduler where the shards are connected in the clique graph with unit distance they provide the stable transaction rate $\rho \leq \max\{\frac{1}{18k}, \frac{1}{\lceil 18\sqrt{s}\rceil}\}$, for which they show the number of pending transactions at any round is at most $4bs$ (which is the upper bound on queue size in each shard), and the latency of transactions is bounded by $36b \cdot \min\{k, \lceil\sqrt{s}\rceil\}$, If this single leader scheduler is considered in the general graph where the transaction and its accessing object are d far away, then their latency becomes $36bd \cdot \min\{k, \lceil\sqrt{s}\rceil\}$. Similarly, for a multi-leader scheduler, they provide a stable transaction injection rate $\rho \leq \frac{1}{c_1' d \log^2 s} \cdot \max\{\frac{1}{k}, \frac{1}{\sqrt{s}}\}$, where c_1' is some positive constant. For this scheduling algorithm, they show the upper bound on queue size as $4bs$, and transaction latency as $2 \cdot c_1' bd \log^2 s \cdot \min\{k, \lceil\sqrt{s}\rceil\}$. However, they consider a synchronous communication model, which is not practical in blockchain, and they also do not provide a theoretical analysis of the optimal approximation for the scheduling algorithm, and they only consider a stateless scheduling model. All their latency bounds depend on the burstiness parameter b, which can be arbitrarily large, as it expresses a transaction injection burst of arbitrary size in any given time interval. On the other hand, our system models do not depend on any burstiness parameter, as we adopt a transaction injection model tuned for performance analysis.

In [3, 14] (see Table 1), the authors presented batch scheduling algorithms (for a given set of transactions) while they did not consider dynamic transaction generation. Moreover, their provided bounds are not tight even for batch processing. Furthermore, their algorithms work on a synchronous communication model, which might not be applicable in a real-world distributed blockchain network. The authors in [14] only consider single leader algorithms and have worse performance complexity bounds than [3] by a factor of $\log D$, resulting in a complexity of $O(kd \cdot \log D)$ whereas [3] achieves $O(kd)$ approximation for batch transactions. Here, we provide efficient scheduling algorithms with theoretical analysis for dynamic transaction processing in a blockchain sharding system that works in the partially synchronous communication model.

Several works have been conducted on transaction scheduling in shared memory multi-core systems, distributed systems, and transactional memory [7, 8]. In [5, 18, 19], the authors explored transaction scheduling in distributed transactional memory systems aimed to achieve better performance bounds with low communication costs. In [7] they provide offline scheduling for transactional memory, where each transaction attempts to access an object, and once it obtains the object, it executes the transaction. In another work [8], the authors extended their analysis from offline to online scheduling for the transactional memory in a synchronous communication model. However, these works do not address transaction scheduling problems in the context of blockchain sharding. This is because, in the transactional memory model, the considered system models assume that objects are mobile, and once a transaction obtains the object, it immediately executes the transaction. In contrast, in blockchain sharding, an object is static in a shard, and there is a confirmation scheme to confirm and commit each subtransaction consistently in the respective shard.

3 Technical Preliminaries

3.1 Blockchain Sharding Model

We consider a blockchain sharding model similar to [11, 1, 2, 3], consisting of n nodes which are partitioned into s shards $S_1, S_2, \ldots, S_s$ such that $S_i \subseteq \{1, \ldots, n\}$, for $i \neq j$, $S_i \cap S_j = \emptyset$, $n = \sum_i |S_i|$, and $n_i = |S_i|$ denotes the number of nodes in shard S_i. Let $G_s = (V, E, w)$ denote a weighed graph of shards, where $V = \{S_1, S_2, \ldots, S_s\}$, the edges E correspond to the connections between the shards, and the weight function w represents the distance between the shards. The graph G_s is complete, since each pair of shards can communicate directly, but the weights of the edges may be non-uniform.

Each shard maintains a local blockchain (which is part of the global blockchain) according to its local accounts and the subtransactions it receives and commits. We use f_i to represent the number of Byzantine nodes in shard S_i. To guarantee consensus on the current state of the local blockchain, we assume that every shard executes the PBFT [9] or a similar consensus algorithm. In order to achieve Byzantine fault tolerance, we assume each shard S_i consists of $n_i > 3f_i$ nodes.

We assume that shards communicate with each other via message passing [11], and here, we are not focusing on optimizing the message size. We adopt the cluster-sending protocol described in [12] and Byshard [11], where shards run consensus (e.g., the PBFT [9] consensus algorithm within the shard) before sending a message. For communication between shards S_1 and S_2, a set $A_1 \subseteq S_1$ of $f_1 + 1$ nodes in S_1 and a set $A_2 \subseteq S_2$ of $f_2 + 1$ nodes in S_2 are chosen (where f_i is the number of faulty nodes in shard S_i). Each node in A_1 is instructed to broadcast the message to all nodes in A_2. Thus, at least one non-faulty node in S_1 will send

the correct message value to a non-faulty node in S_2. (Actually, A_1 needs to have size $2f_1+1$ to distinguish the correct message.) We consider a partial-synchronous communication model, where sending messages for transactions to their accessing shards has a bounded delay.

Suppose we have a set of shared accounts $\mathcal{O}$ (which we also call *objects*). Similar to previous works in [11, 1, 2], we assume that each shard is responsible for a specific subset of the shared objects (accounts). To be more specific, $\mathcal{O}$ is split into disjoint subsets $\mathcal{O}_1, \ldots, \mathcal{O}_s$, where the set of accounts under the control of shard S_i is represented by $\mathcal{O}_i$. Every shard S_i keeps track of local subtransactions that use its corresponding objects in $\mathcal{O}_i$.

3.2 Transactions and Subtransactions

Similar to the works in [11, 2, 3], we consider transactions $\{T_1, T_2, \ldots\}$ that are distributed across different shards. Suppose that transaction T_i is generated in a node v_{T_i} within the system, then the *home shard* of T_i is the shard containing v_{T_i}. In this work, we consider transactions that are continuously generated over time. For simplicity and to attain a performance analysis, we assume that each home shard contains at most one transaction at any moment of time, and after the transaction gets processed (either commits or aborts), a new transaction is generated on that home shard.

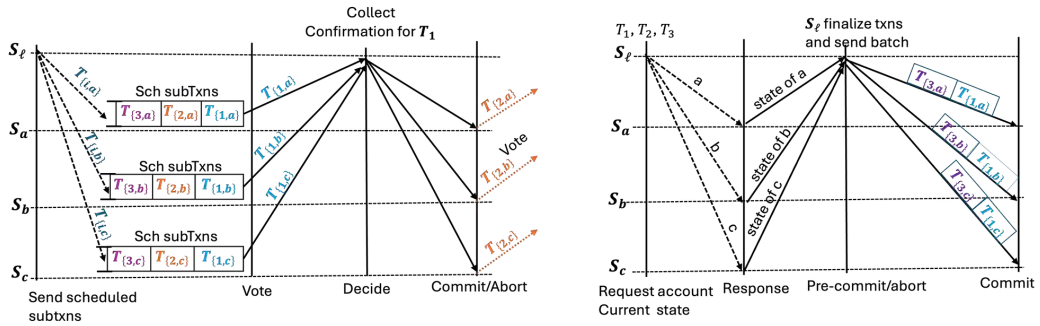
Similar to work in [11, 1, 2], we define a transaction T_i as a group of subtransactions $T_{i,a_1}, \ldots, T_{i,a_j}$. Each subtransaction T_{i,a_l} accesses objects only in $\mathcal{O}_{a_l}$ and is associated with shard S_{a_l}. Therefore, each subtransaction T_{i,a_l} has a respective *destination shard* S_{a_l}. The home shard sends the transaction T_i to the leader shard S_ℓ, which is responsible for processing transaction T_i. Then the leader shard of T_i sends subtransaction T_{i,a_l} to shard S_{a_l} for processing, where it is appended to the local blockchain of S_{a_l}. The subtransactions within a transaction T_i are independent, meaning they do not conflict and can be processed concurrently. An example of transactions and subtransactions is deferred to Appendix A.1.

3.3 Stateless and Stateful Scheduling Models

We define two scheduling models to schedule and process the transactions, the stateless and stateful models, which we describe as follows.

Stateless Scheduling Model. Let's suppose there is a designated *leader shard* S_ℓ that coordinates the scheduling and processing of transactions. In this model, the leader shard S_ℓ does not maintain the current state of accounts accessed by the transactions [2, 3, 11]. Upon receiving transactions, S_ℓ constructs (or extends) a transaction conflict graph and colors the graph using an incremental greedy vertex coloring algorithm to determine the commit order for each transaction. Then the leader S_ℓ splits each transaction into subtransactions based on accessed accounts and sends them to the corresponding *destination shards* that hold the relevant account states. Each destination shard maintains the scheduled subtransactions queue sch_{dq} and it picks one color subtransaction from the head of sch_{dq}, validates the sub-transactions (e.g., checking account balances) and sends a *commit* or *abort* vote to the leader. After collecting all votes for a transaction, the leader sends a final decision to each destination shard, which either commits or aborts the subtransactions according to the message received from the leader shard.

For example, suppose S_ℓ receives transactions T_1, T_2, T_3, each accessing accounts a, b, c, located in shards S_a, S_b, S_c respectively (see Figure 1 (a)). The leader constructs a conflict graph $G_{\mathcal{T}_\ell}$ and applies a greedy vertex coloring algorithm to define a commit order. It then



(a) Stateless Scheduling Model (b) Stateful Scheduling Model

Figure 1 Illustration of Stateless (a) and Stateful (b) Scheduling Models.

splits transactions into sub-transactions:

$$T_1 \to \{T_{1,a}, T_{1,b}, T_{1,c}\}, \quad T_2 \to \{T_{2,a}, T_{2,b}, T_{2,c}\}, \quad T_3 \to \{T_{3,a}, T_{3,b}, T_{3,c}\}$$

Each destination shard queues the received sub-transactions in a schedule queue sch_{dq} according to the commit order received from S_ℓ, and it processes one color subtransaction at a time. This means S_a picks $T_{1,a}$, S_b picks $T_{1,b}$ and S_c picks $T_{1,c}$ from head of their queues, check the validity and condition of the subtransaction (such as account balance) and send either commit or abort votes to the leader shard. Then the transaction T_i and its subtransactions ($T_{1,a}, T_{1,b}$ and $T_{1,c}$) are committed or aborted based on the final decision received from the leader shard. Next, each destination shard processes the next color subtransactions, for instance $T_{2,a}$, from S_a, $T_{2,b}$ from S_b, and $T_{2,c}$ from S_c (see Figure 1 (a)), and this process repeats.

Stateful Scheduling Model. In the stateful model, the home shard where a transaction is initially generated sends its transaction information to the leader shard S_ℓ. Then the leader shard S_ℓ stores these transactions (i.e. $\mathcal{T}_1, T_2, T_3$) in its pending transaction queue PQ_ℓ. Then, the leader shard identifies accounts accessed by transactions and requests their state from corresponding shards S_a, S_b, S_c. In other words, before processing the transactions, the leader collects the current state of all accessed accounts from the corresponding destination shards. Once the account states are gathered, the leader constructs a conflict graph on which it applies the incremental greedy vertex coloring algorithm. Then the leader shard performs local *pre-commit* for valid transactions (e.g., T_1, T_3) and aborts invalid transactions (e.g., T_2). After that, S_ℓ creates the pre-committed sub-transaction batches: $S_a : \{T_{1,a}, T_{3,a}\}$, $S_b : \{T_{1,b}, T_{3,b}\}$, $S_c : \{T_{1,c}, T_{3,c}\}$ for each destination shard S_a, S_b, S_c. Then these pre-committed batches are sent to the respective destination shards. Since the transactions have already been validated, each destination shard can directly commit and append the received pre-committed order to its local blockchain without further interaction with the leader.

The main difference between the stateless and stateful model is that the stateful model requires the leader to be updated about account states which are at remote shards, while the stateless model does not require leader to be informed about remote accounts.

3.4 Conflicts and Competitive Ratio

Two transactions conflict if they attempt to access the same account, and at least one of the two updates the account. The subtransactions are processed sequentially at each destination shard. For this reason, we extend the notion of conflict to all transactions that access account in the same destination shard.

▶ **Definition 1** (Conflict). *Transactions T_i and T_j are said to* conflict *if they access accounts on the same destination shard S_k and at least one of these transactions writes (updates) the account in S_k.*

Transactions that conflict should be processed in a sequential manner to guarantee atomic object update. In such a case, their respective subtransactions should be serialized in the exact same order in every involved shards. To resolve the conflict between two transactions T_i and T_j while accessing the same destination shard S_k, a scheduler must schedule them one after another in such a way that T_i commits before T_j or vice versa. To perform the schedule, we use a *conflict graph* such that the nodes are transactions, and an edge represents a conflict between two transactions.

We continue with the definition of competitive ratio for our scheduling models. The definition below is an adaptation of the competitive ratio used in dynamic execution in software transactional memory [8]. Since the future transactions depend on the past execution, we define the competitive ratio based on any set of transactions that may appear at any moment of time. Consider a transaction schedule S. Let $\mathcal{T}_t$ denote the set of all pending transactions (that have not committed or aborted) at time t. Let $t' > t$ denote the time such that all transactions in $\mathcal{T}_t$ finalize (commit or abort). Let τ^* denote the optimal time duration to finalize (commit or abort) all the transactions in $\mathcal{T}_t$ if they were the only transactions in the system, processed as a batch. The approximation ratio for S at time t is $r_S(t) = (t' - t)/\tau^*$. The competitive ratio for S is $r_S = \sup_t r_S(t)$.

▶ **Definition 2** (Algorithm Competitive Ratio). *For online scheduling algorithm $\mathcal{A}$, the competitive ratio $r_{\mathcal{A}}$ is the maximum competitive ratio over all possible schedules $\mathcal{S}$ that it produces, $r_{\mathcal{A}} = \sup_{S \in \mathcal{S}} r_S$. (We also say that $\mathcal{A}$ is $r_{\mathcal{A}}$-competitive.)*

4 Stateless Scheduler

In this section, we consider the stateless sharding model [1, 2, 11], where the leader shard is responsible for coordinating transaction processing and does not gather the current state of account information (see Section 3.3). We present two transaction scheduling algorithms: the Single-Leader Scheduler and the Multi-Leader Scheduler.

4.1 Stateless Single-Leader Scheduler

In this section, we describe and analyze the *Stateless Single-Leader Scheduler*, which operates under a partially synchronous communication model. We assume a designated leader shard S_ℓ responsible for determining the transaction schedule. All shards send their transactions to the leader shard, which builds a transaction conflict graph and applies an incremental greedy vertex coloring algorithm to determine a schedule.

The algorithm follows an event-driven approach to schedule and process the transactions. When a new transaction T_i is generated at its home shard S_i, then the home shard tags the current timestamp to the transaction T_i and sends the transaction to the leader shard S_ℓ. Upon receiving T_i, the leader adds it to the local transaction set $\mathcal{T}_\ell$ and extends the

Algorithm 1 Stateless Single Leader Scheduler.

txn: transaction; txns: transactions; subTxn: subtransaction; subTxns: subtransactions;
T_i: txn, $T_{i,j}$: subTxn of T_i for shard S_j, $\mathcal{T}_\ell$: Set of txns maintained by leader shard S_ℓ; is_busy: processing flag (initially false at each shard); Each shard S_j maintains a lexicographically ordered scheduled queue sch_{dq} for subtransactions;

Upon generation of a new txn T_i at home shard S_i
  S_i tags local timestamp (ts) to T_i and send it to the leader shard S_ℓ;

Upon receiving new txn T_i at leader shard S_ℓ
  S_ℓ adds T_i to txns set $\mathcal{T}_\ell$ and extend transaction conflict graph $G_{\mathcal{T}_\ell}$ with T_i;
  If any colored txn T_x exists with ts(T_x) > ts(T_i), cancel its color, prioritize T_i, and send cancel message for T_x to corresponding destination shards;
  Run incremental greedy coloring on $G_{\mathcal{T}_\ell}$ without altering already scheduled (colored) old txns;
  Split each newly colored T_i into subtxns $T_{i,j}$ and send to respective destination shard S_j;

Upon receiving subtransaction $T_{i,j}$ at each destination shard S_j
  Append $T_{i,j}$ in sch_{dq} and order (sort) sch_{qd} lexicographically according to color;
  if $is_busy == false$ **then**
    Set $is_busy = true$;
    Let $T_{i,j} \leftarrow$ head of sch_{dq}; If $T_{i,j}$ is valid and local conditions satisfied, it sends *commit vote* to leader shard S_ℓ; Otherwise, it sends *abort vote* to S_ℓ;

Upon receiving votes for txn T_i at leader shard S_ℓ
  If any abort vote receive for T_i then it sends *confirmed abort* to all corresponding S_j of T_i; else if all received votes are commit votes, then it sends *confirmed commit* to corresponding S_j;
  Remove T_i from $\mathcal{T}_\ell$ and $G_{\mathcal{T}_\ell}$ and send outcome(committed or aborted) to home shard of T_i;

Upon receiving confirmation for subtxn $T_{i,j}$ at each destination shard S_j
  If the confirmed commit is received, then it commit $T_{i,j}$ and append to its local blockchain;
  Otherwise, if confirmed abort message received then it abort $T_{i,j}$;
  If sch_{dq} is not empty, it start to process next subTxn from sch_{dq}, else it set $is_busy = false$;

Upon receiving cancel message for $T_{x,j}$ at destination shard S_j
  Remove $T_{x,j}$ from sch_{dq}; a new color will be received later for $T_{x,j}$ from leader shard S_ℓ;

Upon receiving outcome of T_i at home shard S_i
  Generate next transaction and repeat process;

conflict graph $G_{\mathcal{T}_\ell}$ with this new transaction (T_i). If T_i is older than any already-colored but uncommitted transactions (say T_x), the leader cancels the color of those newer transactions, notifies the relevant shards, and reprocesses them later. This ensures older transactions are prioritized, avoiding starvation. The leader then runs an incremental greedy vertex coloring algorithm [8] to assign colors to all newly received transactions, without modifying the colors of already scheduled old transactions. This ensures that the processing time of already scheduled transactions is not affected by newly generated transactions. Note that a newer transaction might receive a lower color than an older one because the new one does not conflict with any other transaction (except one old transaction), while the old transaction conflicts with others as well. To prevent this and ensure a fair execution order, we assign each new transaction a color no lower than the smallest color among pending old transactions. This approach guarantees progress because at each time step, the lowest possible color will increase over time. After coloring and determining the schedule, each transaction is then split into subtransactions $T_{i,j}$ based on the destination shards it accesses, and these subtransactions are sent to the corresponding shards S_j for processing.

Each destination shard S_j maintains a local scheduled queue sch_{dq} (consisting of subtransactions that have been scheduled but not yet committed) and appends incoming subtransactions into sch_{dq}, which stores subtransactions in the order of their assigned color. To handle partial synchrony, each destination shard S_j uses a busy flag to track whether it is currently processing (in-transit and not committed yet) a subtransaction. If the shard is not busy, it picks one subtransaction from the head of the queue and validates it (e.g., checking conditions like account balance). If the subtransaction is valid, the shard S_j sends a *commit*

vote to the leader S_ℓ; otherwise, it sends an *abort vote*. Once the leader shard receives votes from all relevant destination shards for a transaction T_i, it decides whether the transaction should be committed or aborted. If all subtransactions vote to commit, the leader sends a *confirmed commit* to each destination shard; otherwise, if any one of the shard send an abort vote, it sends a *confirmed abort*. After the decision, the transaction T_i is removed from the conflict graph $G_{\mathcal{T}_\ell}$ and the transaction set $\mathcal{T}_\ell$, and the outcome (committed or aborted) is sent to the home shard of T_i.

Upon receiving the confirmed decision, each destination shard either commits the subtransaction by appending it to the local blockchain or aborts it. If the scheduled queue is not empty, the shard continues processing the next subtransaction. If the queue becomes empty, the shard marks itself as not busy. Finally, upon receiving the outcome from the leader, the home shard generates a new transaction and repeats the process. This single leader scheduling approach ensures conflict-free execution while preserving consistency and fairness in transaction processing across shards.

Performance Analysis of Single-Leader Scheduler (Algorithm 1). Our proposed scheduling algorithm works on a partial-synchronous communication model; for the sake of performance analysis only, we consider the synchronous communication mode. In the following, we analyze the time units required to process transactions by Algorithm 1. We are focusing on the time period after the leader shard has determined the schedule for the transactions. In the synchronous case, a time unit is the time to send a message along an edge of unit weight. In the single-leader case, d is sensitive to the position of the leader and d denotes the maximum distance between any of the involved shards (home, destination shards, leader shard). In the multi-leader case, the distance to the leaders is not included in the definition of d.

▶ **Theorem 3.** *[General Graph] In the General graph, where the transactions, their accessing objects, and the leader are at most d distance away from each other, Algorithm 1 has $O(d \cdot \min\{k, \sqrt{s}\})$ competitive ratio.*

Proof. Consider a set of transactions $\mathcal{T}$ generated at or before time t that are still pending (neither committed nor aborted) at time t. Let $G_{\mathcal{T}}$ denote the conflict graph for $\mathcal{T}$, where two transactions conflict if they have a common destination shard. Since we use greedy coloring to color $G_{\mathcal{T}}$, the number of distinct colors assigned to the transactions in $\mathcal{T}$ depends only on the coloring of $G_{\mathcal{T}}$, and not on the colors of the transactions that have been finalized (committed or aborted) before t. (This holds since transactions in $\mathcal{T}$ may use smaller colors of transactions committed before t.)

Let l_i denote the number of transactions in $\mathcal{T}$ that use objects in shard S_i. Let $l = \max l_i$. We have that l is a lower bound on the time that it takes to finalize (commit or abort) the transactions in $\mathcal{T}$, since at least l subtransactions need to serialize in a destination shard.

First, consider the case where $k \leq \sqrt{s}$. We have that each transaction conflicts with at most kl other transactions. Hence $G_{\mathcal{T}}$ can be colored with at most $kl+1$ colors. The distance between a transaction (home shard) and its accessing objects(destination shards) is at most d away, and to commit subtransactions after being scheduled, Algorithm 1 takes 3 steps of interactions (for each color) between the leader shard and the destination shard. This means each color corresponds to the $3d$ time units. Thus, it takes at most $(kl+1)3d = O(kld)$ time units to confirm and commit the transactions. Hence, for transactions $\mathcal{T}$, the approximation of their finalization time is $O(kld/l) = O(kd)$.

Next, consider the case $k > \sqrt{s}$. We can write $\mathcal{T}' = A \cup B$, where A are the transactions which access at most $\sqrt{s}$ destination shards, while B are the transactions which access more than $\sqrt{s}$ destination shards. Each transaction in A conflicts with at most $l\sqrt{s}$ other

transactions. Hence, the transactions in A need at most $l\sqrt{s}+1$ distinct colors. The transactions in B can be serialized, requiring at most $|B|$ distinct colors. Hence, the conflict graph G_T can be colored with at most $l\sqrt{s}+1+|B|$ colors, which implies a schedule of length $O(d(l\sqrt{s}+|B|))$ steps to finalize the transactions $\mathcal{T}$. Since each transaction in B accesses more than $\sqrt{s}$ shards, there is a shard accessed by more than $(|B|\sqrt{s})/s = |B|/\sqrt{s}$ transactions. Thus, $l > |B|/\sqrt{s}$. Hence, for transactions $\mathcal{T}$, the approximation of their finalization time is $O(d(l\sqrt{s}+|B|)/l) = O(d\sqrt{s}+d|B|/l) = O(d\sqrt{s}+d\sqrt{s}) = O(d\sqrt{s})$.

Therefore, combining the approximations for the cases $k \leq \sqrt{s}$ and $k > \sqrt{s}$, we have that the combined approximation for the finalization time for $\mathcal{T}$ is $O(d \cdot \min\{k, \sqrt{s}\})$. Since t is chosen arbitrarily, we have that the competitive ratio of Algorithm 1 is $O(d \cdot \min\{k, \sqrt{s}\})$. ◀

Suppose that shards are connected in a clique graph with unit distance, where every shard is connected to every other shard with unit distance. So in this case $d = 1$. Then from Theorem 3, Algorithm 1 has an $O(\min\{k, \sqrt{s}\})$ competitive ratio for a clique graph with unit distance. Thus, we have:

▶ **Corollary 4** (Unit Distance Clique). *Algorithm 1 has an $O(\min\{k, \sqrt{s}\})$ competitive ratio for a clique graph with unit distance.*

We continue to show that it is an NP-hard problem to approximate the optimal transaction schedule. Thus, the provided bound in Corollary 4, is the best we can do with a polynomial time scheduling algorithm. The result below applies to both the stateful and stateless model.

▶ **Theorem 5.** *For all $\epsilon > 0$, it is an NP-hard problem to produce a transaction schedule that achieves a competitive ratio $(\min\{k, \sqrt{s}\})^{1-\epsilon}$.*

The proof of Theorem 5 is deferred to Appendix A.3.

4.2 Multi-Leader Scheduler

This section provides the multi-leader scheduler where multiple leaders schedule and process the transactions, distribute the congestion, and load across different leaders. The multi-leader approach allows adaptation to the value d without requiring knowledge of d. Also, here the value d depends only on the maximum distance between the home and destination shards (without involving distances to the leaders). Therefore, the value of d captures better the locality of the transactions, and the resulting schedule allows for shorter messages between home and destination shards. The concepts that we introduce for this algorithm will play a key role for the development of the stateless multi-leader algorithm.

4.2.1 Shard Clustering

In the multi-leader scheduler, shards are distributed across the network, and the distance between the home shard of the transaction and its accessing objects (destination shards) ranges from 1 to D, where D is the diameter of the shard graph. Let us suppose shards graph G_s constructed with s shards, where the weights of edges between shards denote the distances between them. We consider that G_s is known to all the shards. We define z-neighborhood of shard S_i as the set of shards within a distance of at most z from S_i. Moreover, the 0-neighborhood of shard S_i is the S_i itself.

We consider that our multi-leader scheduling algorithm uses a hierarchical decomposition of G_s which is known to all the shards and calculated before the algorithm starts. This shard clustering (graph decomposition) is based on the clustering techniques in [10] and which

were later used in [18, 8, 2]. We divide the shard graph G_s into the hierarchy of clusters with $H_1 = \lceil \log D \rceil + 1$ layers (logarithms are in base 2), and a layer is a set of clusters, and a cluster is a set of shards. Layer q, where $0 \leq q < H_1$, is a sparse cover of G_s such that: (i) Every cluster of layer q has (strong) diameter of at most $O(2^q \log s)$. (ii) Every shard participates in no more than $O(\log s)$ different clusters at layer q. (iii) For each shard S_i there exists a cluster at layer q which contains the $(2^q - 1)$-neighborhood of S_i within that cluster.

For each layer q, the sparse cover construction in [10] is actually obtained as a collection of $H_2 = O(\log s)$ partitions of G_s. These H_2 partitions are ordered as sub-layers of layer q labeled from 0 to $H_2 - 1$. A shard might participate in all H_2 sub-layers but potentially belongs to a different cluster at each sub-layer. At least one of these H_2 clusters at layer q contains the whole $2^q - 1$ neighborhood of S_i.

In each cluster at layer q, a leader shard S_ℓ is specifically designated such that the leader's $(2^q - 1)$-neighborhood is in that cluster. As we give an idea of layers and sub-layers, we define the concept of height as a tuple $h = (h_1, h_2)$, where h_1 denotes the layer and h_2 denotes the sub-layer. Similar to [18, 8, 2], heights follow lexicographic order.

The *home cluster* for each transaction T_i is defined as follows: suppose S_i is the home shard of T_i, and z is the maximum distance from S_i to the destination shards that will be accessed by T_i; the home cluster of T_i is the lowest-layer (and lowest sub-layer) cluster in the hierarchy that contains z-neighborhood of S_i. Each home cluster consists of one dedicated leader shard, which will handle all the transactions that have their home shard in that cluster (i.e., transaction information will be sent from the home shard to the cluster leader shard to determine the schedule). An example of hierarchical clustering is presented in Appendix A.4.

4.2.2 Stateless Multi-Leader Scheduler

We consider a hierarchical clustering of the shard graph G_s, which is assumed to be globally known by all shards. Each cluster C in this hierarchy is characterized by a unique height (q, r) which corresponds to its layer q and sublayer r, and each cluster C has its designated leader shard S_ℓ. The leader shard is responsible for scheduling and coordinating the processing of all transactions whose home cluster is C. Each home shard S_i maintains a local timestamp ts to tag newly generated transactions. Additionally, each destination shard S_j maintains a local scheduling queue sch_{dq} and lexicographically orders for the incoming subtransactions using the tuple $(ts, q, r, color)$, where $color$ is an integer assigned to the transaction by the leader shard S_ℓ through vertex coloring. Algorithm 2 invokes Algorithm 1 in each cluster C to process their transactions.

Algorithm 2 works in a partially synchronous model and follows an event-driven execution by message passing. When a new transaction T_i is generated at its home shard S_i, then the home shard S_i determines the lowest cluster C at height (q, r) that includes both S_i and all of the destination shards accessed by T_i. Moreover, the transaction is tagged with its local timestamp ts, along with the cluster identifiers q and r, and is then sent to the cluster's leader shard S_ℓ.

Upon receiving new transaction(s) T_i, the leader shard S_ℓ of cluster C invokes Algorithm 1 to process their transactions, where leader shard S_ℓ adds T_i to the transaction set $\mathcal{T}_C$ of cluster C and updates the corresponding transaction conflict graph $G_{\mathcal{T}_C}$ to incorporate the new transaction T_i. Then the leader shard uses an incremental greedy vertex coloring algorithm [8] to assign a color only to the newly received transaction without affecting already colored (scheduled) transactions. Once colored, the transaction is split into subtransactions $T_{i,j}$, and sent to the respective destination shard S_j.

Algorithm 2 Stateless Multi-Leader Scheduler.

Assume all shards know a hierarchical cluster decomposition of G_s;
Each cluster C is associated with a unique height (q, r) and has a designated leader shard S_ℓ;
Each shard S_j maintains a lexicographically ordered queue sch_{dq} for subtransactions;
Upon generation of a new transaction T_i at home shard S_i
    S_i tags a local timestamp (ts) to T_i and identifies the destination shards accessed by T_i;
    S_i selects the lowest cluster C with height (q, r) that contains T_i and all its destination shards;
    S_i sends T_i to the leader shard S_ℓ of cluster C;
Upon receiving transaction T_i at the leader shard S_ℓ of cluster C
    The leader shard S_ℓ of each cluster C invokes Algorithm 1 to schedule and process their transactions. This means each cluster C invokes Algorithm 1 to process their transactions;
    `// Since multiple clusters process their transactions concurrently, each with its own leader, destination shards may receive subtransactions from different clusters simultaneously.`
    To handle subtransactions from multiple clusters (leaders):
        Each destination shard S_j maintains a scheduled subtransactions queue sch_{dq} ordered lexicographically by the tuple $(ts, q, r, color)$. The additional parameters (ts, q, r) reflect the hierarchical cluster heights (layers and sublayers) in the shard graph G_s;
        Each destination shard S_j processes their subtransactions from the head of sch_{dq} following the rules in Algorithm 1, with the modified ordering criteria;

Since multiple leader shards process their transactions concurrently by invoking the Algorithm 1, destination shards may receive the subtransactions from different clusters simultaneously. To handle this, we modify the parameters and processing technique of Algorithm 1 as follows: each destination shard S_j maintains a scheduled subtransactions queue sch_{qd}, which is ordered lexicographically by the tuple $(ts, q, r, color)$. The additional parameters (ts, q, r) denote the timestamp ts, and hierarchical cluster heights (layers q and sublayers r) in the shard graph G_s. Moreover, each destination shard S_j processes its subtransactions from the head of sch_{dq} following the steps in Algorithm 1 with the modified ordering criteria.

Additionally, if the destination shard is busy and receives a new subtransaction $T_{i',j}$ such that $ts(T_{i',j}) < ts(T_{i,j})$ in lexicographic order, this means $T_{i',j}$ has a higher priority where $T_{i,j}$ is the currently processed (but not committed) subtransaction, then the shard give priority to $T_{i',j}$ by sending an *ignore* $T_{i,j}$ message to its leader, indicating that a higher-priority transaction (subtransaction $T_{i',j}$) should proceed first. Then, when the leader receives an *ignore* $T_{i,j}$ message for a subtransaction $T_{i,j}$ and the decision for T_i has not yet been made (i.e., not all votes have been received), the leader discards the vote from S_j and replies with an *ignored* $T_{i,j}$ message to the destination shard S_j. If the decision has already been made (i.e, confirm commit or confirm abort) by the leader shard, then no further action is taken for particular subtransaction $T_{i,j}$ at the leader shard S_ℓ. After the destination shard S_j receives an *ignored* message for $T_{i,j}$, then it reinserts $T_{i,j}$ into the scheduled queue, reorders the queue lexicographically, and resumes processing from the head.

Finally, when the home shard S_i receives the final outcome of its transaction T_i, it generates a new transaction and sends it to the corresponding cluster leader shard, and the process repeats. This multi-leader scheduling framework ensures conflict-free and consistent execution by leveraging lexicographic ordering over the tuple $(ts, q, r, color)$, and maintains the fairness and parallelism across shards in the presence of partial synchrony.

Performance Analysis of Stateless Multi-Leader Scheduler. The multi-leader scheduler is the extended version of the single-leader scheduler (Algorithm 3) while introducing an additional overhead cost due to its shard (hierarchical) clustering structure and comes from the layers and sublayers of the clusters.

▶ **Theorem 6.** *In Multi-leader scheduler (Algorithm 2), where the transactions and their accessing objects are at most d distance away from each other, Algorithm 2 has $O(d \log^2 s \cdot \min\{k, \sqrt{s}\})$ competitive ratio.*

The proof of Theorem 6 is deferred to Appendix A.6.

5 Stateful Scheduler

In this scheduler model, the leader shard gathers all of the transactions and the current states of the accessing accounts and pre-commits the transactions at the leader. After that, the leader creates the pre-committed subtransactions batch and sends that batch to the respective destination shard, where each destination shard reaches a consensus on the received subtransaction order and adds it to their local blockchain. We provide two stateful scheduling algorithms, one with a single leader and the other with multiple leaders.

5.1 Stateful Single-Leader Scheduler

We present and analyze the *stateful single-leader scheduler*, where one of the shards is considered as the leader S_ℓ, which is responsible for scheduling and processing all the transactions.

When a new transaction T_i is generated at its home shard S_i, S_i sends T_i to the leader shard S_ℓ. Upon receipt, S_ℓ appends T_i to its local pending queue PQ_ℓ. Scheduling event is triggered periodically, either every 4λ time units or upon processing transactions associated with λ distinct colors. Here, λ denotes the worst-case communication delay between any two shards, which is at most the diameter of the shard communication graph G_s. The 4λ bound accounts for the communication delays involved in acquiring state information from remote shards and completing the pre-commitment phase and sending the pre-committing batch to the destination shard.

When the scheduling event is triggered, the leader shard moves its pending transactions from PQ_ℓ into the scheduling transaction set $\mathcal{T}_\ell$ and identifies the set of accounts $\mathcal{O}_v$ accessed by transactions which are in T_ℓ. If any account state $O_j \in \mathcal{O}_v$ is not locally available at S_ℓ, it determines the responsible destination shard S_j for each such account, and sends batched account state requests to the corresponding shards. If all required states are already available in S_ℓ, an internal State-Ready (i.e. already available locally) event is triggered immediately.

Upon receiving a state request, each destination shard S_j responds with the current state of the requested accounts (e.g., balances). Then, once all necessary account states are collected at S_ℓ, it extends the conflict graph $G_{\mathcal{T}_\ell}$ by incorporating the new transactions in $\mathcal{T}_\ell$ Then the leader shard S_ℓ runs the incremental greedy vertex coloring algorithm [8] on $G_{\mathcal{T}_\ell}$ and assigns at most ζ colors without altering the coloring of previously scheduled old transactions.

The leader then iteratively processes transactions color by color. For each color group ζ_c, S_ℓ verifies transaction conditions (e.g., sufficient balance) using the up-to-date account state it gathers. Transactions that are valid and conditions are satisfied are *pre-committed*, while invalid ones are aborted. Then S_ℓ splits each pre-committed transaction T_i into subtransactions $T_{i,j}$ based on its accessed shards. These subtransactions are appended to a corresponding pre-commit batch $PrecommitSubTxnBatch(S_j)$ for each destination shard S_j. After processing a transaction, it is removed from $\mathcal{T}_\ell$ and $G_{\mathcal{T}_\ell}$, and the outcome (committed or aborted) is reported back to the transaction's home shard S_i to initiate the next transaction.

Algorithm 3 Steteful Single Leader Scheduler.

S_ℓ: Leader shard; PQ_ℓ: Pending txns queue in leader shard;
$\mathcal{T}_\ell$: Set of scheduled txns maintained by leader; $G_{\mathcal{T}_\ell}$: Conflict txn graph on $\mathcal{T}_\ell$;
λ: worst communication delay between any two shards due to partial-synchrony;
$PrecommitSubTxnBatch(S_j)$: Precommitted subtransactions batch for shard S_j;

Upon generation of a new txn T_i **at home shard** S_i
  S_i sends T_i to the leader shard S_ℓ;

Upon receiving new txn T_i **at leader shard** S_ℓ
  S_ℓ appends T_i to PQ_ℓ;
  if S_ℓ *waits for* 4λ *time unit or* S_ℓ *proceed* λ *number of scheduled colors* **then**
    `// Trigger scheduling event`
    Move txns from PQ_ℓ to $\mathcal{T}_\ell$; Identify set of accessed accounts $\mathcal{O}_v$ by txns in $\mathcal{T}_\ell$;
    if *Current state of any account* $O_j \in \mathcal{O}_v$ *is not locally available at* S_ℓ **then**
      For each O_j, determine the responsible shard S_j and create request batch for each S_j;
      Send batched account state request to each destination shard S_j;
    else
      S_ℓ has all accounts state, so trigger internal state-ready event (see below);

Upon receiving a batched account state request at destination shard S_j
  Respond to leader shard S_ℓ with current states of all requested accounts;

Upon receiving account states from each S_j**, or already available locally at** S_ℓ
  S_ℓ extend txn conflict graph $G_{\mathcal{T}_\ell}$ with new txns in $\mathcal{T}_\ell$ and runs incremental greedy vertex coloring algorithm on $G_{\mathcal{T}_\ell}$ using ζ colors without altering already scheduled old txns;
  foreach *color* $\zeta_c \in \zeta$ **do**
    Pre-commit or abort txns $T_i \in \zeta_c$ by checking txn condition and account state;
    If T_i is pre-committed, split T_i into subTxns and create (append) pre-committed subtxns batch order $PrecommitSubTxnBatch(S_j)$ for each destination shard S_j;
    Remove T_i from $\mathcal{T}_\ell$ and $G_{\mathcal{T}_\ell}$. Send the outcome(committed/aborted) to home shard of T_i;
    `// Track processed color`
    if *processed* λ *number of colors* **then**
      break;
  S_ℓ sends $PrecommitSubTxnBatch(S_j)$ to corresponding destination shard S_j parallelly and start to process next batch;

Upon receiving precommitted batch $PrecommitSubTxnBatch(S_j)$ **at each** S_j
  Reach consensus on $PrecommitSubTxnBatch(S_j)$ and append batch to the local blockchain;

The pre-commitment phase terminates once λ colors are processed, after which S_ℓ dispatches all $PrecommitSubTxnBatch(S_j)$ batches to their respective destination shards in parallel. Each destination shard S_j then reaches consensus on the order of subtransactions in the received batch and appends them to its local blockchain. The leader shard S_ℓ, then waits and proceeds to the next scheduling batch.

Performance Analysis of Stateful Single-Leader Scheduler. In the following, we analyze the time unit required to process transactions by Algorithm 3. We focus on the special case where the maximum distance between the transactions, their accessing objects, and the leader is at most d, and at least one transaction accesses objects at a distance $\Omega(d)$. This special case is useful for the analysis of the multi-leader case. We are focusing on the time period after the leader shard has determined the schedule for the transactions. This is because the scheduling and committing steps are executed in parallel.

▶ **Theorem 7.** *[General Graph] In the General graph, where the transactions, their accessing objects, and the leader are at most d distance away from each other, and at least one transaction is $\Omega(d)$ distance from the accessing shards, Algorithm 3 has $O(\min\{k, \sqrt{s}\})$ competitive ratio.*

The proof of Theorem 7 is deferred to Appendix A.7.

5.2 Stateful Multi-Leader Scheduler

We present a *stateful multi-leader scheduler* in which multiple leader shards are responsible for scheduling and processing transactions. In the single-leader algorithm, the value d includes the distance to the leader, but in the multi-leader, d does not include the relative distance to the leader. This allows the multi-leader algorithm to capture better the locality of transactions, allowing for shorter distance messages between the involved home and destination shards.

The system assumes a hierarchical cluster decomposition [10] of the shard graph G_s, which is globally known to all shards. Each cluster $C(q, r)$ in the hierarchy is associated with a leader shard S_ℓ, a pending transaction queue PQ_ℓ, a scheduled transaction set $\mathcal{T}_\ell$, and a transaction conflict graph $G_{\mathcal{T}_\ell}$. The parameter λ_C denotes the worst-case communication delay between any two shards within the cluster C, which arises from the assumption of a partially synchronous communication model.

In multi-leader scheduling Algorithm 4 (see in Appendix A.9), when a transaction T_i is generated at its home shard S_i, the shard identifies the lowest cluster $C(q, r)$ that contains all the shards accessed by T_i, and then forwards T_i to the leader shard S_ℓ of of cluster C. The leader shard S_ℓ appends the received transaction to its pending queue PQ_ℓ. Periodically, the leader checks if either $4\lambda_C$ time units have elapsed since the last scheduling event or if λ_C colors of scheduled transactions have been processed by S_ℓ. If either condition is met and the leader holds the `scheduleControl`, it invokes the single-leader scheduler (Algorithm 3) on its local structures $(PQ_\ell, \mathcal{T}_\ell, G_{\mathcal{T}_\ell}, \lambda_C)$ to process transactions.

The scheduling control, denoted by the boolean flag `scheduleControl`, determines which cluster can perform scheduling operations at a given time unit. The control flows hierarchically between parent and child clusters. A parent cluster of C is any cluster at a higher level in the hierarchy (with height $(q', r') > (q, r)$) that shares at least one shard with C. Similarly, a child cluster of C is a lower-level cluster (with height $(q'', r'') < (q, r)$) that overlaps with C. Clusters may have multiple parents and children. If C is at the bottom-most level (height $(0, 0)$), initially it has `scheduleControl`. Otherwise, it must request control from all its children. Once all children respond the `scheduleControl`, the leader S_ℓ sets `scheduleControl` to true and proceeds with the scheduling.

After executing the single-leader scheduler, if the parent cluster C' requests control, the leader transfers `scheduleControl` to the parent and sets it to false locally. If instead a child cluster C'' has made a request, the control is passed down to the child. If there are no remaining transactions to process, the control is also passed downward to allow lower-level clusters to schedule pending transactions. If the leader does not have `scheduleControl` when scheduling should occur, it sends a control request to the current holder (parent or child). Additionally, if C receives a control request from a parent C' while not holding control, it forwards the request to its children. Once all children respond positively, it passes control up to C'. This hierarchical and event-driven mechanism ensures coordinated and conflict-free scheduling across multiple levels of the cluster hierarchy.

▶ **Theorem 8.** *In Multi-leader scheduler (Algorithm 4), where the transactions and their accessing objects are at most d distance away from each other, Algorithm 4 has $O(\log s \cdot \min\{k, \sqrt{s}\} + \log^2 s)$ competitive ratio.*

The proof of Theorem 8 is deferred to Appendix A.8.

6 Conclusion

We presented efficient scheduling algorithms for processing dynamic transactions in blockchain sharding systems. Our proposed framework operates under a partially synchronous communication model, which realistically captures the behavior of many real-world blockchain environments. We introduced both stateless and stateful scheduling models, each of which includes single-leader and multi-leader algorithms for transaction scheduling and processing. For these algorithms, we provided competitive ratios relative to an optimal scheduler and established both upper and lower bounds on the scheduling delay.

For future work, we plan to explore efficient inter-shard communication mechanisms, particularly under conditions of network congestion where communication links have bounded capacity. We also aim to conduct extensive simulations and real-world experiments to evaluate the practical performance of our proposed protocols.

References

1 Ramesh Adhikari and Costas Busch. Lockless blockchain sharding with multiversion control. In *International Colloquium on Structural Information and Communication Complexity*, pages 112–131. Springer, 2023. doi:10.1007/978-3-031-32733-9_6.

2 Ramesh Adhikari, Costas Busch, and Dariusz Kowalski. Stable blockchain sharding under adversarial transaction generation. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*, 2024.

3 Ramesh Adhikari, Costas Busch, and Miroslav Popovic. Fast transaction scheduling in blockchain sharding. *arXiv preprint arXiv:2405.15015*, 2024. doi:10.48550/arXiv.2405.15015.

4 Ramesh Adhikari, Costas Busch, and Miroslav Popovic. On the efficiency of dynamic transaction scheduling in blockchain sharding. *arXiv preprint arXiv:2508.07472*, 2025.

5 Hagit Attiya, Vincent Gramoli, and Alessia Milani. Directory protocols for distributed transactional memory. *Transactional Memory. Foundations, Algorithms, Tools, and Applications: COST Action Euro-TM IC1001*, pages 367–391, 2015. doi:10.1007/978-3-319-14720-8_17.

6 Zeta Avarikioti and Dimitris Karakostas. Harmony technical report, 2022.

7 Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. Fast scheduling in distributed transactional memory. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 173–182, 2017. doi:10.1145/3087556.3087565.

8 Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. Dynamic scheduling in distributed transactional memory. *Distributed Computing*, 35(1):19–36, 2022. doi:10.1007/S00446-021-00410-W.

9 Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

10 Anupam Gupta, Mohammad T Hajiaghayi, and Harald Räcke. Oblivious network design. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 970–979, 2006.

11 Jelle Hellings and Mohammad Sadoghi. Byshard: Sharding in a byzantine environment. *Proceedings of the VLDB Endowment*, 14(11):2230–2243, 2021. doi:10.14778/3476249.3476275.

12 Jelle Hellings and Mohammad Sadoghi. The fault-tolerant cluster-sending problem. In *Foundations of Information and Knowledge Systems: 12th International Symposium, FoIKS 2022, Helsinki, Finland, June 20–23, 2022, Proceedings*, pages 168–186. Springer, 2022. doi:10.1007/978-3-031-11321-5_10.

13 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018. doi:10.1109/SP.2018.000-5.

14 Ao Liu, Jing Chen, Kun He, Ruiying Du, Jiahua Xu, Cong Wu, Yebo Feng, Teng Li, and Jianfeng Ma. Dynashard: Secure and adaptive blockchain sharding protocol with hybrid consensus and dynamic shard management. *IEEE Internet of Things Journal*, 2024.

15 Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 17–30, 2016. doi:10.1145/2976749.2978389.

16 Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. A survey of blockchain from the perspectives of applications, challenges, and opportunities. *Ieee Access*, 7:117134–117151, 2019. doi:10.1109/ACCESS.2019.2936094.

17 A Secure. The zilliqa project: A secure, scalable blockchain platform, 2018.

18 Gokarna Sharma and Costas Busch. Distributed transactional memory for general networks. *Distributed computing*, 27(5):329–362, 2014. doi:10.1007/S00446-014-0214-7.

19 Gokarna Sharma and Costas Busch. A load balanced directory for distributed shared memory objects. *Journal of Parallel and Distributed Computing*, 78:6–24, 2015. doi:10.1016/J.JPDC.2015.02.002.

20 Alex Skidanov and Illia Polosukhin. Nightshade: Near protocol sharding design. *URL: https://nearprotocol. com/downloads/Nightshade. pdf*, 39, 2019.

21 Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 931–948, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3243734.3243853.

22 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 681–690, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132612.

A Appendix

A.1 Example of Transaction and Subtransactions

Suppose transaction T_1 is: *"Transfer 100 coins from account A to account B"*. Let us assume that the accounts of A and B reside on different shards S_a and S_b, respectively. T_1 splits into the following subtransactions:

$T_{1,a}$ in S_a: Condition: Check if account A has at least 100 coins.
Action: Deduct 100 coins from account A.
$T_{1,b}$ in S_b: Action: Add 100 coins to account B.

A.2 Correctness Analysis of Stateless Single-Leader Scheduler

Our proposed scheduling algorithm works on a partial-synchronous communication model; for the sake of analysis only, we consider the synchronous communication mode.

▶ **Lemma 9** (Safety). *If two transactions conflict with each other in Algorithm 1, then they will commit in different time slots, and the local chain produced by Algorithm 1 ensures blockchain serialization.*

Proof. We prove this by induction (analyzing) the execution of Algorithm 1, where each home shard sends its transaction to the leader shard (Line 4), and the leader shard constructs the transaction conflict graph $G_{\mathcal{T}_\ell}$ (Line 6). Then the leader used the incremental greedy vertex coloring algorithm [8] on the conflict graph $G_{\mathcal{T}_\ell}$ (Line 8). As conflicting transactions

share an edge in $G_{\mathcal{T}_\ell}$, they are assigned different colors and are processed in different time slots, which provides the valid commit order. Moreover, each color corresponds to a unique serialization time slot. The leader shard splits the transaction into subtransactions and sends them to the destination shard after coloring (see Line 9), then each destination shard keeps that ordering in the schedule queue (sch_{dq}) and process subtransactions one by one according to the color they get (see Line 11-14), which guarantees the consistent schedule order in each shard. Moreover, the leader shard coordinates to commit the subtransactions in each destination shard, which ensures the consistent commitment (see Line 16-17). As the subtransactions are committed according to the color they receive, and each color corresponds to a globally consistent time slot, this provides global serialization. ◀

▶ **Lemma 10** (Liveness). *Algorithm 1 guarantees that every generated transaction will eventually be either committed or aborted.*

Proof. We prove liveness by induction, showing that every transaction T_i is either committed or aborted in finite time. Each new transaction T_i is sent to a leader shard S_ℓ (Line 4), which adds it to the set $\mathcal{T}_\ell$ and the conflict graph $G_{\mathcal{T}_\ell}$. If T_i is older than any already colored but not committed transaction T_x, the algorithm cancels the color of T_x and re-colors the graph (Line 7). Coloring is performed incrementally (Line 8) and preserves the colors of previously scheduled transactions. Thus, older transactions are always prioritized, and no transaction is indefinitely prevented from being scheduled due to newer ones. Note that a newer transaction might receive a lower color than an older one because the new one does not conflict with any other transaction (except one old transaction), while the old transaction conflicts with others as well. To prevent this and ensure a fair execution order, we assign each new transaction a color no lower than the smallest color among pending old transactions. This approach guarantees progress because at each time step, the lowest possible color will increase over time.

Moreover, once T_i is colored, its subtransactions are sent to the respective destination shards (Line 9), where they are placed into a queue sch_{dq} sorted by color (Line 11). Each shard processes one color group at a time, controlled by a busy flag. After finishing one subtransaction (commit or abort), the shard proceeds to the next one in the queue. Since every color is eventually dequeued, and subtransactions are processed in order, every scheduled subtransaction is eventually processed. Thus, every transaction is either committed or aborted in a finite time, and this proves the liveness. ◀

▶ **Corollary 11.** *From Lemma 9 and Lemma 10, Algorithm 1 ensures the safety and liveness of the transactions.*

A.3 Proof of Theorem 5

Proof. We will use a reduction from vertex coloring. For all $\epsilon > 0$, the problem of approximating the chromatic number of a graph with n nodes within a factor $n^{1-\epsilon}$ is NP-hard [22].

Consider an instance of vertex coloring on a graph $H = (V_H, E_H)$ with n nodes. We can transform the vertex coloring instance H to a scheduling problem instance on a graph shard G_s with $s = |E_H|$ shards, such that G_s is a synchronous clique with unit distances between the shards. Furthermore, each edge of E_H corresponds to a unique node of G_s.

Let $\mathcal{T}$ be a set of n transactions, all generated concurrently at time $t = 0$, such that each node $v_i \in V_H$ is mapped to transaction $T_i \in \mathcal{T}$. For each edge $(v_i, v_j) \in E_H$ we create a conflict between respective transactions T_i and T_j by making the transactions access a common object in the unique shard of G_s that corresponds to edge (v_i, v_j). Let $G_{\mathcal{T}}$ be the respective conflict graph for the transactions $\mathcal{T}$. The conflict graph $G_{\mathcal{T}}$ is isomorphic to H.

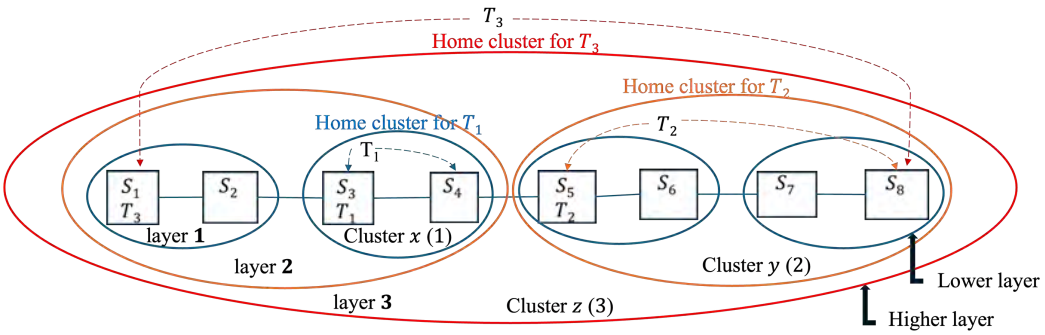


Figure 2 Simple example of cluster decomposition of shard graph G_s.

A correct execution schedule for $\mathcal{T}$ (which gives a valid serialization of the transactions in $\mathcal{T}$) can be represented as a DAG where nodes are transactions and transaction T_i points to T_j if they conflict and T_i executes first in the respective common destination shard with T_j. Then, a layering of the DAG nodes starting from source nodes provides a unique time step for each transaction, so that conflicting transactions receive different time steps. Thus, an execution schedule of the transactions in $\mathcal{T}$ gives a valid vertex coloring of the nodes in $G_{\mathcal{T}}$ which provides a valid coloring for H. The best length of the transaction schedule given from the DAG, is equal to the number of colors that can be assigned to H.

Since $|E_H| \leq n(n-1)/2$, we have that $s = O(n^2)$. Each transaction conflicts with at most $k \leq n-1$ other transactions. Therefore, given k and s, we can create the reduction from graph coloring for $n = \min(k, \sqrt{s})$. Consequently, the NP-hardness of the scheduling problem in G_s follows from the NP-hardness of the reduced graph coloring problem with $n = \min(k, \sqrt{s})$. ◀

A.4 Example of Hierarchical Clustering

Figure 2 shows an example of hierarchical clustering, assuming shards are connected as if they are in a line, where edges in the line have low weights and edges not in the line have large weights. (We omit the sublayers to simplify the example.) Transaction T_1 resides in shard S_3 and has home cluster x at layer 1. The reason for the home cluster x selection is that T_1 accesses an object in S_3 and S_4, and both of them are in cluster x, and x is the lowest layer cluster including S_3 and S_4. Similarly, suppose transaction T_2, which resides in S_5, has home cluster y at layer 2, because T_2 accesses an object in S_5 and S_8, and y is the lowest layer cluster that includes both S_5 and S_8. Similarly, T_3 has home cluster z at layer 3.

A.5 Correctness Analysis of Stateless Multi-Leader Scheduler

▶ **Lemma 12** (Safety). *If two transactions conflict with each other in Algorithm 2, then they will commit in different time slots, and the local chain produced by Algorithm 2 ensures blockchain serialization.*

The proof of Lemma 12 is available in arxiv [4]

▶ **Lemma 13** (Liveness). *Algorithm 2 guarantees that every generated transaction will eventually be committed or aborted.*

The proof of Lemma 13 is available in arxiv [4].

▶ **Corollary 14.** *From Lemma 12 and Lemma 13, Algorithm 2 ensures the safety and liveness of the transactions.*

A.6 Proof of Theorem 6

Proof. In the multi-layer scheduler, we need to consider the transactions from all layers and sublayers of the clusters. Suppose q' is the topmost layer accessed by any transaction where the diameter of the cluster on that layer is at most $d_{q'}$.

Consider the destination shard S_j, and we have only subtransactions from one leader shard of cluster layer q where the distance between the transaction and its accessing shard is at most d_q, and it has maximum competitive ratio denoted by $\tau_q = O(d_q \cdot \min\{k, \sqrt{s}\})$ (from Theorem 3) than any other cluster. Therefore the destination shard S_j needs to process subtransactions from all layers $0, \ldots, q'$ and from sublayers $0, \ldots, H_2 - 1$, and those transactions are processed according to their assigned order.

As discussed in Section 4.2.1, a cluster at layer q has a diameter at most $O(2^q \log s)$. Thus $d_q = O(2^q \log s) = c2^q \log s$, for some positive constant c. This implies $\sum_{q=0}^{q'} d_q \leq 2d_{q'}$. Thus, the competitive ratio of Algorithm 2 considering transactions from all layers and sublayers at destination shard S_j is at most:

$$\tau_{total} \leq \sum_{q=0}^{q'} \sum_{r=0}^{H_2-1} \tau_q \leq \sum_{q=0}^{q'} \sum_{r=0}^{H_2-1} O(d_q \cdot \min\{k, \sqrt{s}\}) \leq O(d_{q'} H_2 \cdot min\{k, \sqrt{s}\}) \ . \tag{1}$$

We can replace $H_2 = O(\log s)$ and $d_{q'} = O(d \log s)$ (see Section 4.2.1), then Equation 1 becomes:

$$O(d \log^2 s \cdot min\{k, \sqrt{s}\}) \ . \qquad \blacktriangleleft$$

The correctness analysis of stateful single leader and multi-leader scheduler is available in arxiv [4].

A.7 Proof of Theorem 7

Proof. This proof follows the same arguments discussed in the proof of Theorem 3. Consider a set of transactions $\mathcal{T}$ generated at or before time t that are still pending (neither committed nor aborted) at time t. Let $G_{\mathcal{T}}$ denote the conflict graph for $\mathcal{T}$, where two transactions conflict if they have a common destination shard. Let l_i denote the number of transactions in $\mathcal{T}$ that use objects in shard S_i. Let $l = \max l_i$. Moreover, from the definition of d, at least one transaction is d distance away from the destination shard or leader. So we have that $\Omega(l + d)$ is a lower bound on the time that it takes to finalize (commit or abort) the transactions in $\mathcal{T}$, since at least l subtransactions need to serialize in a destination shard, and at least one transaction is d distance away.

First, consider the case where $k \leq \sqrt{s}$. We have that each transaction conflicts with at most kl other transactions. Hence $G_{\mathcal{T}}$ can be colored with at most $kl + 1$ colors.

Algorithm 3 schedules and commits transactions in batches. For each batch, the leader shard performs the following steps: first, it gathers the state of accessed accounts, takes at most $2d$ time units (request and receive each takes at most d time units). After pre-committing, the leader sends the pre-commit batch to destination shards, which takes d time units. Additionally, destination shards reach consensus on the received batch within 1 time unit. Hence, the total delay per batch is at most $3d + 1$.

Since the algorithm uses at most $kl + 1$ colors (batches), the total finalization time is at most: $kl + 1 + 3d + 2 = O(kl + d)$.

Next, consider the case $k > \sqrt{s}$. Following the same reasoning above and from Theorem 3, we get $O(l\sqrt{s} + d)$ time to finalize the transactions $\mathcal{T}$.

Overall, Algorithm 3 requires $O(l \cdot \min\{k, \sqrt{s}\} + d)$ time units to finalize the transactions. Since $\Omega(l + d)$ is a lower bound, we have that the approximation factor of the schedule for $\mathcal{T}$ is $O(\min\{k, \sqrt{s}\})$.

Since t is chosen arbitrarily, we have that the competitive ratio of Algorithm 3 is $O(\min\{k, \sqrt{s}\})$. ◀

A.8 Proof of Theorem 8

Proof. Similar to Theorem 6, consider the destination shard S_j, as discussed in the proof of Theorem 7, if we have only subtransactions from one leader shard of cluster layer q where the distance between the transaction and its accessing shard is at most d_q, then the time to process transactions is $O(l \cdot \min\{k, \sqrt{s}\} + d_q)$ or equivalently at most $c_1(l \cdot \min\{k, \sqrt{s}\} + d_q)$ time for some positive constant c_1. Suppose q' is the maximum layer accessed by any transaction where the diameter of the cluster on that layer is at most $d_{q'}$. Then the destination shard S_j needs to process subtransactions from all layers $0, \ldots, q'$ and from sublayers $0, \ldots, H_2 - 1$, and those transactions are processed according to their assigned order.

As discussed in Section 4.2.1, a cluster at layer q has a diameter at most $O(2^q \log s)$. Thus $d_q = O(2^q \log s) = c2^q \log s$, for some positive constant c. This implies $\sum_{q=0}^{q'} d_q \leq 2d_{q'}$. Thus, the total time unit required by Algorithm 4 to process all the transactions from all layer and sublayers at destination shard S_j is at most:

$$\tau_{total} \leq \sum_{q=0}^{q'} \sum_{r=0}^{H_2-1} c_1(l \cdot \min\{k, \sqrt{s}\} + d_q) \leq c_1 l H_2 \cdot min\{k, \sqrt{s}\} + 2c_1 d_{q'} H_2 \ . \tag{2}$$

We can replace $H_2 = c_2 \log s$ as we have $O(\log s)$ sublayers (see Section 4.2.1) and $d_{q'} = c_3 d \log s$, where c_2 and c_3 are some positive constants, then Equation 2 becomes:

$$c_1 l \cdot c_2 \log s \cdot min\{k, \sqrt{s}\} + 2c_1 \cdot c_3 d \log s \cdot c_2 \log s => O(l \log s \cdot min\{k, \sqrt{s}\} + d \log^2 s) \ .$$

As discussed in Theorem 7, $\Omega(l + d)$ is a lower bound. Thus, we have that the competitive ratio of Algorithm 4 as $O(\log s \cdot \min\{k, \sqrt{s}\} + \log^2 s)$. ◀

A.9 Pseudocode of Stateful Multi-Leader Scheduler

Algorithm 4 Stateful Multi-Leader Scheduler.

1 Each shard knows the hierarchical cluster decomposition of G_s;
2 Each cluster $C(q,r)$ has: leader shard S_ℓ, txn queue PQ_ℓ, scheduled txns $\mathcal{T}_\ell$, conflict graph $G_{\mathcal{T}_\ell}$;
3 λ_C: worst communication delay between any two shards in cluster C due to partial-synchrony;
4 `scheduleControl`: Boolean flag indicating whether the cluster currently holds scheduling control;
5 **Upon generation of new txn** T_i **at home shard** S_i
6 S_i determines the lowest cluster $C(q,r)$ which includes T_i and its accessing shards. Then S_i sends T_i to leader shard S_ℓ of $C(q,r)$;
7 **Upon receiving txn(s)** T_i **at leader shard** S_ℓ **of** $C(q,r)$
8 S_ℓ appends T_i to its pending transactions queue PQ_ℓ;
9 **if** S_ℓ *waits for* $4\lambda_C$ *time unit or* S_ℓ *proceed* λ_C *number of previous scheduled colors* **then**
10 **if** `scheduleControl` $==$ *True* **then**
11 `// Invoke single-leader scheduling logic`
12 Run Single-Leader Scheduler (Algorithm 3) with $(PQ_\ell, \mathcal{T}_\ell, G_{\mathcal{T}_\ell}, \lambda_C)$;
13 `// If Algorithm 3 break after process` λ_C `number of scheduled colors then check and do following:`
14 **if** *parent cluster* C' *requests control* **then**
15 Send `scheduleControl` to the parent and set `scheduleControl` $\leftarrow$ False;
16 **else if** *children clusters* C'' *request control* **then**
17 Send `scheduleControl` to children and set `scheduleControl` $\leftarrow$ False;
18 **else if** $C(q,r)$ *doesn't have remaining transactions to schedule* **then**
19 Send `scheduleControl` down to children and set `scheduleControl` $\leftarrow$ False;
20 **else**
21 Send request to current `scheduleControl` holder (e.g., child or parent cluster);
22 **Upon receiving** `scheduleControl` **at leader** S_ℓ **of** $C(q,r)$
23 **if** S_ℓ *previously requested scheduledControl to process its txns* **then**
24 Set `scheduleControl` $\leftarrow$ True and trigger internal event (see above on line 9-12);
25 **else**
26 Send `scheduleControl` to parent or child clusters according to the request it gets;

An Almost-Logarithmic Lower Bound for Leader Election with Bounded Value Contention

Dan Alistarh
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Faith Ellen
University of Toronto, Canada

Alexander Fedorov
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

We investigate the step complexity of the Leader Election problem (and implementing the corresponding test-and-set object) in asynchronous shared memory, where processes communicate through registers supporting atomic read and write and must coordinate so that a single process becomes the leader. Determining tight step complexity bounds for solving this problem is one of the key open problems in the theory of shared memory distributed computing. The best known algorithm is a randomized tournament-tree, which has worst-case expected step complexity $O(\log N)$ for N processes. There are provably no deterministic wait-free algorithms, and only restricted lower bounds are known for obstruction-free and randomized wait-free algorithms. We introduce a new lower bound that establishes an $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ step complexity for any obstruction-free Leader Election algorithm, where N is the number of processes, and $2 \leq Q \leq N$ is a bound on the *value contention*, which we define as the maximum number of different values that processes can be simultaneously poised to write to the same register in any execution of the algorithm. Our result is strictly stronger than previous bounds based on write contention. In particular, it implies new lower bounds on step complexity that depend on register size.

2012 ACM Subject Classification Theory of computation → Shared memory algorithms; Theory of computation → Lower bounds and information complexity; Theory of computation → Concurrency

Keywords and phrases Leader Election, Test-and-Set, Shared Memory, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.3

Funding The work of Dan Alistarh is supported by grants from ERC, Austrian FWF, and the Google and NVIDIA corporations. Faith Ellen was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC) grant RGPIN-2020-04178.

Acknowledgements The authors would like to thank the DISC 2025 anonymous reviewers for their detailed comments and, in particular, Reviewer C, who inspired our approach for a new and simpler proof of Theorem 5 via the probabilistic method.

1 Introduction

Leader Election is a fundamental coordination problem in distributed computing, in which, as the name suggests, a set of n processes compete to become the leader. Each process should output either `win` or `lose`, indicating whether the process became the leader. At most one process should become the leader and, if all processes that compete take enough steps, then exactly one of these processes becomes the leader. Note that it is not required for other processes to know the identity of the leader, as this would make the problem equivalent to *Consensus*, which is known to be a harder coordination problem.


© Dan Alistarh, Faith Ellen, and Alexander Fedorov;
licensed under Creative Commons License CC-BY 4.0



A closely-related problem is implementing a *test-and-set* object. It differs from Leader Election in that no process may return *lose* before the eventual winner has started the computation. However, any Leader Election algorithm can be deterministically transformed into an implementation of a test-and-set object with just one additional register and at most two additional steps by each process [4].

Leader Election and test-and-set objects have been studied with different formulations and in various models for almost four decades, e.g. [25, 28, 2, 12, 5]. In this paper, we will focus on Leader Election in the *asynchronous shared memory* model [22, 9], where processes can communicate only through *registers*, shared objects that support *read* and *write* operations. Read returns the value stored in the register, while a write atomically update the stored value. In this way, one process can learn about another process if it reads a register that was last written to by that process. A process executes reads and writes on arbitrary registers one by one, but the way in which the operations of a process are interleaved with those of other processes is controlled by an *adversary*. Specifically, it is assumed that the adversary decides the order of the steps, with the goal of making processes execute as many steps as possible. It is known that a test-and-set object has consensus number two. Thus, it has no deterministic wait-free implementation from registers [17].

1.1 Prior Work

The first randomized algorithm for Leader Election among two processes was given by Tromp and Vitányi [26]. The approach was later generalized to N processes by Afek, Gafni, Tromp and Vitányi [1] using a tournament tree [24], in which each process starts at a leaf of the tree. At each node, the process competes in an instance of Leader Election among two processes. If it loses, it immediately returns *lose*. If it wins, it continues to the parent node. The winner at the root can return *win*. This algorithm has $O(\log N)$ worst-case expected step complexity, since each instance of Leader Election among two processes takes an expected constant number of steps and the tournament tree has $O(\log N)$ depth. This can be straightforwardly modified to an obstruction-free variant with logarithmic solo step complexity [5].

The approach described above leads to the best known upper bound for the case of a strong adversary. Whether this upper bound is optimal remains an open problem [15]. Later work extended these results to the adaptive setting, where the step complexity depends on the number of participating processes, k. Specifically, Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui proposed the `RatRace` algorithm which achieves $O(\log k)$ step complexity in expectation [4, 14]. Golab, Hendler and Woelfel designed an algorithm that solves Leader Election in $O(1)$ remote memory accesses [16]. However, their algorithm is not obstruction-free: if a process crashes, this may cause another process to wait forever for the value in a register to change.

Remarkably, despite decades of interest in the relationship between upper and lower bounds for Leader Election in shared memory, we still do not know whether the logarithmic upper bound is tight, neither for the expected step complexity of randomized algorithms against a strong adversary, nor for the solo step complexity of deterministic obstruction-free algorithms. Intuitively, proving lower bounds for test-and-set is hard because of the apparent simplicity of the object.

The list of relevant lower bounds for Leader Election against a strong adversary is shown in Table 1. In 1989, Styer and Peterson proved that any Leader Election algorithm has to use $\Omega(\log N)$ registers [25]. This space complexity lower bound was later matched by an obstruction-free Leader Election algorithm that uses $O(\log N)$ registers [13]. The remaining lower bounds, including ours, build on ideas from the lower bounds for mutual exclusion

established by James H. Anderson and Yong-Jik Kim [7]. They proved an $\Omega(\frac{\log N}{\log \log N})$ step complexity lower bound for Mutual Exclusion by inductively constructing an execution. Their analysis distinguishes between two scenarios: the high-contention and the low-contention cases. In the low-contention case, they apply a graph theory result to find a large set of processes that do not conflict with one another and use them to extend their execution. In the high-contention case, they run some processes until they enter and leave the critical section, thereby overwriting the contents of the highly contended registers.

The first work to apply Anderson and Kim's ideas to Leader Election was by Alistarh, Gelashvili, and Nadiradze [5]. They proved that any algorithm has worst-case expected step complexity $\Omega\left(\frac{\log N}{\log \kappa}\right)$, where κ is the maximum *write contention*. However, this lower bound becomes trivial when polynomially many processes can write to the same register, that is, when $\kappa = N^{\varepsilon}$, for some constant $\varepsilon > 0$. In other related work, Eghbali and Woefel proved an $\Omega(\frac{\log N}{\log \log N})$ lower bound for implementing an *Abortable* Test-And-Set object [10]. They use a different model, which counts the maximum number of remote memory references (RMRs) performed by a process, rather than the maximum number of steps. However, their proof still follows Anderson and Kim's framework. To handle the high-contention case, their proof heavily relies on the fact that processes can be aborted after they overwrite some register. Thus, their results apply only to Abortable Leader Election, and not to our model.

Table 1 Lower bounds for Leader Election against strong adversary.

Lower Bound	Reference	Comment
$\Omega(\log N)$ *space* complexity	[25]	
$\Omega\left(\frac{\log N}{\log \kappa}\right)$ step complexity	[5]	κ is the write contention
$\Omega\left(\frac{\log N}{\log \log N}\right)$ RMR complexity	[10]	Only for *Abortable* Leader Election
$\Omega\left(\frac{\log N}{\log \log N + \log Q}\right)$ step complexity	**This paper**	Q is the value contention

1.2 Contribution

In this paper, we present a new technique that leads to a lower bound for Leader Election under a more general notion of contention, which we call *value contention*. Specifically, we show a lower bound of $\Omega\left(\frac{\log N}{\log \log N + \log Q}\right)$ on the solo step complexity of any obstruction-free Leader Election algorithm for N processes, where $2 \leq Q \leq N$ is an upper bound on the *value contention*, that is, the *number of different values that processes can be simultaneously poised to write to the same register* in any execution.

This result leads to a new trade-off between the maximum register capacity and the solo step complexity of test-and-set: for instance, if registers can only hold a *poly-logarithmic* number of different values, then test-and-set requires $\Omega(\log N / \log \log N)$ steps in a solo execution, in the worst-case. Similarly, if at most polylog N processes may be simultaneously poised to write to the same register, we obtain the $\Omega(\log N / \log \log N)$ lower bound of Alistarh, Gelashvili and Nadiradze [5], but via new argument. Put differently, our proof shows that constant-time algorithms for test-and-set are only possible if, for some constant $0 < c \leq 1$, N^c processes can concurrently be poised to write different values to the same register. Furthermore, our lower bound shows that no adaptive Leader Election algorithm is possible, unless registers can hold polynomial in N many values, and polynomial in N processes can be poised to write different values to the same register at the same time. We say that an algorithm is *adaptive* if its step complexity depends only on the actual number of participating processes and not on the total number of processes in the system.

Our lower bound is based on a technique that generalizes the knowledge-based approach of Anderson and Kim [7]. At a high level, we inductively construct a sequence of executions in which no active process sees any other active process. Specifically, this leads to an execution by one process in which it takes $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ steps. In each round of the execution, active processes perform one additional read or write. Thus, by round t, each has taken t steps. We split these active processes into groups, called *factions*, depending on the step they are about to perform. We then proceed to schedule the processes adversarially, to minimize the information flow between them. Since information flow may arise from values written to registers in previous rounds, we also erase some processes, or even entire factions, from the execution. To determine which active processes should continue to the next round, we build a conflict graph, where an edge exists between two processes if executing them both would create information flow, or if the two processes are poised to write different values to the same register simultaneously.

So far, this construction is similar to step complexity lower bound proofs for low contention [7, 10, 5]. The key new observation is that we can adjust the execution so that the conflict graph can be represented by a new, special kind of structure that we call a *faction graph*. Roughly, a faction graph can be partitioned into sets of vertices with super-edges from vertices to parts of the partition. Intuitively, this allows us to generate a *compact representation* for any conflict graph. Our main technical lemma is a lower bound of $\Omega\left(\frac{|V|^2}{|E^*|+|V|}\right)$ on the maximum size of an independent set in any conflict graph represented by a faction graph with $|V|$ vertices and $|E^\star|$ multi-edges. This lemma may be of separate interest for other problems that can be described using faction graphs.

Our lower bound argument combines the fact that the conflict graph at each round can be represented by a faction graph, with our technical lemma about such graphs to obtain a new lower bound of $\Omega\left(\frac{N}{Q+t^2}\right)$ on the number of processes that survive until the t^{th} round. These processes do not see one another. We can continue, while keeping at least one process active, for $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$ rounds.

The paper by Alistarh, Gelashvili and Nadiradze [5] presents a complex, potential-based covering argument leading to a lower bound of $\Omega(\log N/\log\kappa)$ that depends on step contention κ. In a follow-up online revision [6], the authors provided a simpler argument, which carefully re-works the original Anderson and Kim lower bound to obtain a simpler proof of the same lower bound. Our lower bound proves a slightly weaker result in the bounded *step contention* case. Our focus is on the bounded *value contention* case, which is a generalization of the case when registers have bounded size.

On the upper bound side, we observe that the tournament tree algorithm by Afek, Gafni, Tromp and Vitányi [1] has constant bounded value contention Q. In fact, with slight modifications, it uses only single-bit registers, and thus has value contention $Q = 2$. Therefore, our $\Omega\left(\frac{\log N}{\log\log N}\right)$ lower bound almost matches the $O(\log N)$ upper bound. Moreover, even though there are adaptive Leader Election algorithms with $O(\log k)$ step complexity [4, 14] (where k is the number of participating processes), our main lower bound implies that adaptive Leader Election is impossible unless $Q \in \Omega(N^c)$ for some constant $c > 0$. In other words, we prove that any algorithm for adaptive Leader Election must use registers of logarithmic size and has executions where the number of processes simultaneously poised to write different values to the same register is polynomial in N.

2 Preliminaries

We consider an asynchronous distributed system where a set of N processes, P, communicate through a set of R registers that can be read from and written to by any process. A *step* is a read of a specific register or a write of a specific value to a specific register performed by one process, followed by an update of the state of the process, including any local coin flips it wishes to perform.

An *execution* $\mathcal{E}$ is a sequence of steps of processes in the system, starting from the initial configuration. An execution is P'*-only* if it only contains steps performed by processes in $P' \subseteq P$. If $\mathcal{E}$ is a prefix of execution $\mathcal{E}'$, we say that $\mathcal{E}'$ extends $\mathcal{E}$. A process p is *poised to write* to a register r at the end of an execution $\mathcal{E}$ if, in every execution $\mathcal{E}'$ that extends $\mathcal{E}$ and in which p performs more steps, p's next step after $\mathcal{E}$ is writing to r. An execution is *terminating* if all operations (that have begun) have finished. A *round-by-round execution* is an execution in which, for every positive integer i, all processes that take at least i steps do so before any process takes its $i+1$'st step. The i'th *round* of the execution is a consecutive sequence of steps by different processes, each of which has previously taken $i-1$ steps. Thus, a round-by-round execution is a concatenation of rounds.

The *step complexity* of an execution is the maximum number of steps performed by one process. The *step complexity* of an algorithm is the maximum step complexity of any execution of the algorithm. The *solo step complexity* of an algorithm is the maximum number of steps in any solo execution of the algorithm, i.e., in which only one process takes steps. The *total step complexity* of an algorithm is the maximum number of steps in any execution of the algorithm.

Adversary. This paper assumes the strong (adaptive) adversary model. In this model, the order in which processes take steps is decided adaptively by an adversary. It knows everything about the current state of all processes when it chooses which process takes the next step. By comparison, weaker adversaries have only partial information about process states such as the outcomes of random coin flips [3, 14]. To be specific, our lower bound proofs assume that the adversary knows the next step of every process, but does not need other information about internal process states.

Worst-case expected step complexity. Consider a (possibly randomized) algorithm $\mathcal{A}$ and an adaptive adversary D, which we assume, for simplicity, to be deterministic. Let $T_{p,D}$ be the random variable denoting the number of steps process p performs in the execution of $\mathcal{A}$ when the scheduling is done by the adversary D. The *expected step complexity* of an execution under D is $\mathbb{E}[T] = \max_{p \in P}[T_{p,D}]$, where the expectation is taken over the processors' coin flips. The *worst-case expected step complexity* of algorithm $\mathcal{A}$ is $\sup_D [\mathbb{E}(T_{p,D})]$, i.e. the supremum over the expected step complexity of $\mathcal{A}$, under any adversary.

Leader Election. A *Leader Election* algorithm supports a single operation, `elect`, which each process may perform at most once. This operation returns either `win` or `lose` indicating whether the process won the election. It satisfies the following properties:

- With probability 1, every process that does not crash finishes its operation in a finite number of steps.
- At most one operation returns `win`.
- If all operations that started have finished, then exactly one operation returned `win`.

Splitter. A *Splitter* [21, 23] is an object that supports a single operation, `decide`, which each process may perform at most once. It satisfies the following properties:

- The output of `decide` is either `Stop`, `Left`, or `Right`.
- Every process that calls `decide` finishes it within a finite number of steps.
- At most one process can get `Stop`.
- If only one process calls `decide`, then it is guaranteed to get `Stop`.
- If more than one process calls `decide`, then not all processes get the same output.

Instead of the last property, a *Randomized Splitter* [8] requires that the probability a call of `decide` returns `Left` is the same as the probability it returns `Right`, independent of all other calls. Splitter and Randomized Splitter objects are similar to Leader Election, where getting `Stop` as output is analogous to winning, but they do not guarantee that some process gets output `Stop`, even if all calls have terminated.

We now describe formalize some well-known results used for to proving lower bounds, which were previously applied for this and other distributed problems [19, 14, 5, 25, 10].

Process Visibility. We say that process p *sees* another process if p reads a value from some register that differs from the last value p wrote there or is not the initial value of the register, if p never wrote there. We say that p *sees* process q if q was the last process that wrote to that register before p read it. For any execution $\mathcal{E}$, define $p \leftrightarrow_{\mathcal{E}} q$ if p sees q or q sees p in execution $\mathcal{E}$. Let $p \sim_{\mathcal{E}} q$ be the reflexive transitive closure of $\leftrightarrow_{\mathcal{E}}$. This is an equivalence relation. A set of processes $P' \subseteq P$ is *closed* (with respect to $\sim_{\mathcal{E}}$) if there does not exist $q \in P'$ and $p \notin P'$ such that $q \sim_{\mathcal{E}} p$.

Note that, if P' is a set of processes that is closed with respect to $\sim_{\mathcal{E}}$, then erasing the steps of all processes in P' from $\mathcal{E}$ results in an execution. Since exactly one operation wins in every terminating execution of Leader Election, we get the following result.

▶ **Lemma 1.** *Let $\mathcal{E}$ be an execution of a Leader Election algorithm and let P' be a nonempty set of processes that is closed with respect to $\sim_{\mathcal{E}}$. Let $\mathcal{E}'$ be an execution obtained by erasing the steps of all processes in $P \setminus P'$ from $\mathcal{E}$ and then running all processes in P' until they each finish performing `elect`. Then exactly one process in P' wins in $\mathcal{E}'$.*

Based on this lemma, we formulate the first commonly used idea guaranteeing that processes have to take more steps in an execution $\mathcal{E}$ of Leader Election that contains at least two processes that are not related by $\sim_{\mathcal{E}}$.

▶ **Lemma 2.** *Let $\mathcal{E}$ be an execution of a Leader Election algorithm. Consider a partition of P into sets that are closed with respect to $\sim_{\mathcal{E}}$. Then every set, except possibly one, contains at least one process that has not finished performing `elect`.*

Proof. For every part, P', of the partition, there is an execution obtained by erasing the steps of all processes in $P \setminus P'$ from $\mathcal{E}$ and then running all processes in P' until they each finish performing `elect()`. By Lemma 1, exactly one process in P' wins in this execution. Consider the winning process for each set P' in the partition. Since at most one process wins in $\mathcal{E}$, each of these processes, except possibly one, has not won in $\mathcal{E}$ and, thus, it has not finished performing `elect`. ◀

Intuitively, Lemma 2 shows that, if we execute processes in a way that prevents them from learning about one another, we can ensure that they have to take more steps.

3 Independent Set Lower Bound for Faction Graphs

In this section, we describe our main combinatorial result, which we later use to prove our main lower bound. It is a stronger, specialized version of Turán's Theorem [27]:

▶ **Theorem 3.** *Every graph $G = (V, E)$ contains an independent set of size at least* $\left\lfloor \frac{|V|^2}{2|E|+|V|} \right\rfloor$.

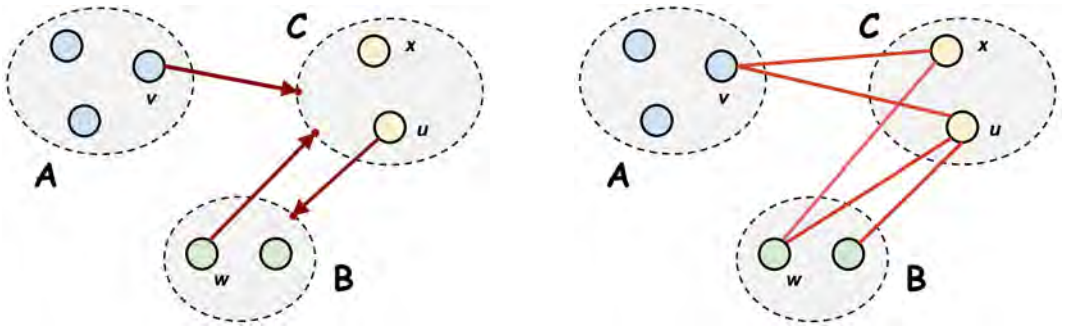
We prove that any graph that can be represented in a special form, which we call a faction graph, has a sufficiently large independent set.

Faction Graphs. A *faction graph* $G^\star = (V, F, E^\star)$ is defined by a partition, F, of its vertex set, V, and a set of *super-edges*, $E^\star$, which connects vertices to parts of the partition, called *factions*. It contains no edges from any vertex to the faction that contains it. The graph $G = (V, E)$ *represented by* $G^\star$ is an undirected graph with the property that, if a vertex v has a super-edge in $E^\star$ to a faction, $A \in F$, then all vertices in A are neighbors of v in G. An example of a faction graph is shown in Figure 1a. The formal definition is given below.

▶ **Definition 4.** *$G^\star = (V, F, E^\star)$ is a* faction graph *if V is a set of vertices, F is a partition of V, and $E^\star \subseteq V \times F$ such that $v \notin A$ for all $(v, A) \in E^\star$. The graph* represented by *this faction graph is $G = (V, E)$, where $E = \{\{v, w\} \mid (v, A) \in E^\star$ and $w \in A$ for some $A \in F\}$.*

Generally, a faction graph is a much more compact representation of the graph it represents.

Figure 1a is a faction graph with 7 vertices, 3 factions, A, B, and C, and 3 super-edges. Figure 1b is the graph it represents. The super-edge (w, C) is expanded into the two edges $\{w, x\}$ and $\{w, u\}$.



(a) A faction graph. **(b)** The graph it represents.

Figure 1 Illustration of faction graphs.

Independent Sets in Faction Graphs. We will show, using the probabilistic method, that the graph $G = (V, E)$ represented by a faction graph $G^\star = (V, F, E^\star)$ has an independent set of size $\Omega\left(\frac{|V|^2}{|E^*|+|V|}\right)$. This is similar to Turán's theorem (Theorem 3). Our key observation is that we can obtain a much bigger lower bound on the size of an independent set, since $|E^*|$ can be asymptotically much smaller than $|E|$.

▶ **Theorem 5.** *The graph $G = (V, E)$ represented by a faction graph $G^\star = (V, F, E^\star)$ contains an independent set of size at least*

$$\frac{1}{e} \cdot \frac{|V|^2}{|V| + |E^\star|} \in \Omega\left(\frac{|V|^2}{|V| + |E^\star|}\right).$$

Proof. Let $N = |V|$ and $M^\star = |E^\star|$. If $E^\star = \emptyset$, then $E = \emptyset$, so V is an independent set of G of size $|V|$. Therefore, assume $M^\star > 0$.

Since there are no edges in E between two nodes in the same faction, every faction is an independent set. For each vertex $v \in V$, let $F(v) \in F$ be the faction that contains v and let

$$\mathrm{OutNhd}^\star(v) = \{\, C \in F \mid (v, C) \in E^\star \,\}.$$

By definition, $F(v) \notin \mathrm{OutNhd}^\star(v)$. Observe that

$$\sum_{v \in V} |\mathrm{OutNhd}^\star(v)| \;=\; |E^\star| \;=\; M^\star,$$

since every super-edge $(v, C) \in E^\star$ contributes exactly once to the sum.

We use the probabilistic method. Select each faction $A \in F$, independently, with some fixed probability $p \in (0, 1)$, obtaining a random subset $F' \subseteq F$. From F' build a random vertex set

$$S = \big\{\, v \in V \mid F(v) \in F' \;\text{ and }\; F' \cap \mathrm{OutNhd}^\star(v) = \emptyset \,\big\}$$

consisting of every vertex whose faction was selected, but has no super-edge to any selected faction.

First, we show that S is an independent set of $G = (V, E)$. Consider any two vertices, $u, v \in S$. If $F(u) = F(v)$, then $\{u, v\} \notin E$, since every faction is an independent set. So suppose $F(u) \neq F(v)$. By definition of S, $F(u), F(v) \in F'$, so $F(v) \notin \mathrm{OutNhd}^\star(u)$ and $F(u) \notin \mathrm{OutNhd}^\star(v)$. Hence $(u, F(v)), (v, F(u)) \notin E^\star$. By Definition 4, it follows the $\{u, v\} \notin E$.

Next, we get a lower bound on the expected size of S. For any fixed $v \in V$, the event that $F(v) \in F'$ is independent of the event that $F' \cap \mathrm{OutNhd}^\star(v) = \emptyset$. Therefore

$$\Pr[v \in S] \;=\; p\,(1-p)^{|\mathrm{OutNhd}^\star(v)|} \text{ and } \mathbb{E}\big[|S|\big] \;=\; \sum_{v \in V} \Pr[v \in S] \;=\; \sum_{v \in V} p\,(1-p)^{|\mathrm{OutNhd}^\star(v)|}.$$

The function $f(x) = (1-p)^x$ is convex for $p \in (0, 1)$. By Jensen's inequality [18],

$$\frac{1}{N} \sum_{v \in V} (1-p)^{|\mathrm{OutNhd}^\star(v)|} \;\geq\; (1-p)^{\frac{1}{N}\sum_{v \in V} |\mathrm{OutNhd}^\star(v)|}.$$

Hence $\mathbb{E}\big[|S|\big] \;\geq\; N\,p\,(1-p)^\alpha$, where $\alpha = M^\star / N$.

We now choose a good value for p. Consider the function $g(p) = p(1-p)^\alpha$ on the interval $[0, 1]$. Since its derivative is $g'(p) = (1-p)^{\alpha-1}\big(1 - (\alpha+1)p\big)$, it follows that g is maximized when $p = 1/(1+\alpha)$. Using this value of p gives

$$\mathbb{E}\big[|S|\big] \;\geq\; N \cdot \frac{1}{1+\alpha} \cdot \Big(1 - \frac{1}{1+\alpha}\Big)^\alpha \;=\; \frac{N}{1+\alpha} \cdot \Big(\frac{\alpha}{1+\alpha}\Big)^\alpha.$$

Since $\ln(1+x) \leq x$ for $x > -1$, we have

$$\Big(\frac{\alpha}{1+\alpha}\Big)^\alpha \;=\; \exp\Big(\alpha \ln \frac{\alpha}{1+\alpha}\Big) \;=\; \exp\Big(-\alpha \ln\big(1 + \tfrac{1}{\alpha}\big)\Big) \;\geq\; \exp(-1) \;=\; \frac{1}{e}.$$

Hence

$$\mathbb{E}\big[|S|\big] \;\geq\; \frac{1}{e} \cdot \frac{N}{1+\alpha} \;=\; \frac{1}{e} \cdot \frac{N^2}{N + M^\star} = \frac{1}{e} \cdot \frac{|V|^2}{|V| + |E^\star|}.$$

Because S is an independent set for every choice of F', it follows that G contains an independent set S of at least this size. ◀

4 The Leader Election Lower Bound

With these preliminaries in place, we now present a new step complexity lower bound for the Leader Election problem (and implementing a test-and-set object) in asynchronous shared memory. To date, no non-trivial lower bound on step complexity for implementing a test-and-set object that depends only on N is known. Our lower bound is no exception: It assumes bounded *value contention* Q, defined as follows.

▶ **Definition 6** (Bounded Value Contention). *An algorithm has* bounded value contention Q *if, at all points during an execution, processes can be poised to write at most Q different values to the same register.*

Discussion. We note that bounded value contention is strictly more general than both write contention and register size. In particular, bounded write contention κ implies value contention at most κ. However, the reverse does not hold because we can allow arbitrary many processes be simultaneously poised to write *the same value* to a register, without exceeding the value contention bound. Likewise, an upper bound, C, on the number of different values that can be written to a register (which is equivalent to a bound on register size) implies value contention at most C. Again, the reverse does not hold, since arbitrarily many different values can be poised to be written to a register in *different* executions, or at *different* configurations during the same execution without exceeding the value contention bound.

To see the intuition why we need this additional restriction on the algorithm to obtain a non-trivial lower bound, consider the following pseudocode:

```
door := {id}
if door != id: // Fast path check
  return Lose // Observed another process, so can lose
// Otherwise perform slow path Leader Election
```

Here, each process first writes its id to a shared register, `door`, and then reads this register. If it reads the id of another process, it immediately loses. Observe that we can add this code at the beginning of any Leader Election algorithm, since the last process to write to `door` will continue to the next part of the algorithm, if it does not crash. The previous step complexity lower bound proofs construct a worst-case execution by repeatedly selecting a set of processes that each perform one more step, one after the other [5, 10, 7]. However, in an execution that begins with a round in which every process takes one step, there is only one process that does not lose when it performs its second step. Consequently, this process will execute the next part of the algorithm by itself. For some algorithms, for example, one that uses a Splitter, this process will also terminate within a constant number of steps. However, in an execution in which no process writes to `door` between the write to `door` and subsequent read of `door` by the same process, all non-faulty processes reach the next part of the algorithm. Our bounded value contention does not allow all processes to be simultaneously poised to write their ids to the same register, so such an algorithm does not contradict our lower bound.

The main result of this section is the following:

▶ **Theorem 7.** *For any obstruction-free Leader Election algorithm for N processes with value contention Q, there exists a solo execution of length $\Omega\left(\frac{\log N}{\log\log N+\log Q}\right)$.*

Proof. We begin by giving an overview of the proof approach. We will construct a round-by-round execution, one round at a time. In each round, we have a set of active processes that each takes one additional step. We maintain the invariant that the processes do not see one another. In other words, for each active process, the execution is indistinguishable from a solo execution. Initially, all processes are active and the execution is empty.

To add an additional round, we consider the next step of all active processes and partition them into groups depending on what register they access, whether they read or write, and what value they read or what value they write. After that, we construct a conflict graph, where the nodes are the active processes and there is an edge between two processes if one of them sees the other in any extension of the current execution in which each active process takes one more step or if they write different values to the same register. We observe that this graph can be represented by a faction graph. Using Theorem 5, we select a sufficiently large independent set of active processes in the conflict graph. All other processes are erased from the execution and then each process in the independent set takes its next step. Similarly to Anderson and Kim [7], this ensures that each process only reads the initial value of a register or the last value that it wrote there. The construction continues until only one process remains.

Let $P_t \subseteq P$ be the set of active processes that participate in the current execution, $\mathcal{E}_t$. In this execution, no process in P_t sees any other process, so, for all processes $p, q \in P_t$, $p \sim_{\mathcal{E}_t} q$ if and only if $p = q$. By Lemma 2, there is at most one process that has terminated at the end of $\mathcal{E}_t$. Each process in P_t that has not terminated has taken exactly t steps in $\mathcal{E}_t$. We also ensure that no two processes write different values to the same register in their i'th step in $\mathcal{E}_t$, for $1 \leq i \leq t$. Initially, $\mathcal{E}_0$ is empty and $P_0 = P$.

To create $\mathcal{E}_{t+1}$, we begin by partitioning the processes in P_t into groups, where two processes are in the same group if, at the end of $\mathcal{E}_t$,

- they are poised to write the same value into the same register,
- they are poised to read the same register and they have never written to that register, or
- they are poised to read the same register and they last wrote the same value to that register.

If there is a process in P_t that is terminated at the end of $\mathcal{E}_t$, it is put in a group by itself. For each register, there are at most Q groups of processes writing different values to the register, there is at most one group of processes that is reading the register, but have never written to it, and there are at most t groups of processes that are reading the register and last wrote to it in one of the previous t rounds.

We need to show that we can choose sufficiently many active processes $P_{t+1} \subseteq P_t$, which each performs one more step, without letting it see any other process. Other active processes will be erased from the execution when creating $\mathcal{E}_{t+1}$. (Note that one of the processes in P_{t+1} may have terminated.) To determine such a set of active processes, we construct a conflict graph. This is a graph with vertex set P_t, where there is an edge between two processes if one of them is poised to read a register from which it could see the other process or both are poised to write different values to the same register. The conflict graph can be represented by a faction graph, where the factions are the groups of processes. We will describe only the set of super-edges, E^*, of this faction graph from which the set of edges of the conflict graph can be deduced. Intuitively, we want to find an independent set in the conflict graph, remove all steps by other processes in $\mathcal{E}_t$, and add a round to the resulting execution to create $\mathcal{E}_{t+1}$.

There are two rules used to construct the super-edges of the faction graph:

`Rule 1.` **Poised to read after a past write.** Consider some process, p, that wrote a value to register r during $\mathcal{E}_t$. Let v be the value that p last wrote to r. Let A be a group of processes that are poised to read register r and last wrote the value $v' \neq v$ to register r or a group of processes that are poised to read register r, but have never written to register r, and $v' \neq v$ is the initial value of register r. In a solo execution, each process in A would read the value v' when it performs this read. Then $(p, A) \in E^*$.

`Rule 2.` **Poised to write different values.** Consider a group of processes, A, poised to write value v to a register r and a group of processes, B, poised to write value $v' \neq v$ to the register r. Then, $(p, B) \in E^*$, for all processes $p \in A$, and $(q, A) \in E^*$, for all processes $q \in B$.

Since all active processes take t steps during $\mathcal{E}_t$, each process p writes to at most t different registers. For each register, r, to which it wrote, $(p, A) \in E^*$ by Rule 1, only if the processes in group A are poised to read r. At most one value is written to r in each round of $\mathcal{E}_t$, so there are at most $t - 1$ groups of processes poised to read r that last wrote a different value to r than p did. There is at most one group of processes poised to read r that have never written to r. Hence, there are at most $t \cdot t = t^2$ groups A for which $(p, A) \in E^*$ by Rule 1.

Suppose $(p, A) \in E^*$ by Rule 2. Then p is poised to write a value v to a register r at the end of $\mathcal{E}_t$ and the processes in A are poised to write a different value to register r at the end of $\mathcal{E}_t$. Since the value contention of the algorithm is at most Q, there are at most $Q - 1$ different groups that are poised to write values other than v to register r at the end of $\mathcal{E}_t$. Hence, there are at most $Q - 1$ groups A for which $(p, A) \in E^*$ by Rule 2.

In total, $|E^*| \leq |P_t| \cdot (t^2 + Q - 1)$. Let P_{t+1} be a maximum independent set in the conflict graph. By Theorem 5,

$$|P_{t+1}| \geq \frac{1}{e} \cdot \frac{|P_t|^2}{|P_t| + |E^*|} \geq \frac{1}{e} \cdot \frac{|P_t|^2}{|P_t| + |P_t| \cdot (t^2 + Q - 1)} = \frac{1}{e} \cdot \frac{|P_t|}{Q + t^2}.$$

Since $P_t \setminus P_{t+1}$ is closed with respect to $\sim_{\mathcal{E}_t}$, removing the steps of all processes in $P_t \setminus P_{t+1}$ from $\mathcal{E}_t$ results in an execution. Let $\mathcal{E}_{t+1}$ be obtained from this execution by appending a round in which all processes in P_{t+1} that are poised to read each take one step and then all processes in P_{t+1} that are poised to write each take one step. Note that, by Rule 2, at most one value is written to each register during the last round of $\mathcal{E}_{t+1}$. Since at most one value is written to each register during each round of $\mathcal{E}_t$, the same is true for $\mathcal{E}_{t+1}$. Furthermore, since each process in P_t that has not terminated takes exactly t steps during $\mathcal{E}_t$, each process in P_{t+1} that has not terminated takes exactly $t + 1$ steps during $\mathcal{E}_{t+1}$.

Suppose there is a process, $q \in P_{t+1}$, in some group, A, that sees another process during $\mathcal{E}_{t+1}$. Since q sees no other processes during $\mathcal{E}_t$, it must be that, during round $t + 1$, q read a value v from register r and, during $\mathcal{E}_t$, either q did not write to r, or q last wrote some value $v' \neq v$ to r. By definition of the groups, the same is true for all processes in group A. Note that all writes to r during round $t + 1$ occur after q read r. Let $p \in P_{t+1}$ be any process that last wrote v to r during $\mathcal{E}_t$. Note that $p \notin A$. By Rule 1, $(p, A) \in E^*$ is an edge in the faction graph. This implies that $\{p, q\}$ is an edge in the conflict graph, which contradicts the fact that P_{t+1} is an independent set. Hence, no process in P_{t+1} sees another process during $\mathcal{E}_{t+1}$.

Thus, the execution $\mathcal{E}_{t+1}$ has all the required properties. We repeatedly construct additional rounds until the remaining set of active processes has size 1. Note that, if $|P_t| > 1$, it is always possible to choose P_{t+1} so that it contains a process that takes $t + 1$ steps during $\mathcal{E}_{t+1}$. Let T be the first number such that $|P_T| = 1$. Then $\mathcal{E}_T$ is a solo execution of length T.

It remains to show that $T \in \Omega\left(\frac{\log N}{\log Q + \log\log N}\right)$. Recall that $|P_{t+1}| \geq \frac{1}{e} \cdot \frac{|P_t|}{Q+t^2}$, so

$$|P_T| \geq \frac{|P_0|}{e^T \prod_{t=0}^{T-1}(Q+t^2)}.$$

Rearranging and using the facts that $|P_0| = N$ and $|P_T| = 1$, we obtain

$$N \leq e^T \prod_{t=0}^{T-1}(Q+t^2) \leq e^T \prod_{t=0}^{T-1}(Q+T^2) = (e(Q+T^2))^T.$$

Hence,

$$T \geq \frac{\log N}{\log(e(Q+T^2))} = \frac{\log N}{\log e + \log(Q+T^2)}.$$

Let $N \geq 4$. Then $\log Q + \log\log N \geq 1$, since $Q \geq 1$. If $T \geq \log N$, then $T \geq \frac{\log N}{\log Q + \log\log N}$. So, assume that $T < \log N$. In this case,

$$T > \frac{\log N}{\log e + \log(Q + (\log N)^2)}.$$

Since $Q, \log N \geq 1$, we have $Q + (\log N)^2) \leq 2Q(\log N)^2$, so $\log e + \log(Q + (\log N)^2) \leq \log e + \log(2Q(\log N)^2) = 1 + \log e + \log Q + 2\log\log N \in O(\log Q + \log\log N)$. Therefore, $T \in \Omega\left(\frac{\log N}{\log Q + \log\log N}\right)$. ◀

This theorem leads to some additional results. First, we observe that our lower bound argument can be applied directly to the more general class of *non-deterministic solo-terminating* algorithms [11], which includes randomized wait-free algorithms for the Leader Election problem.

▶ **Corollary 8.** *Against an adaptive adversary, every randomized wait-free Leader Election algorithm for N processes with value contention Q has $\Omega\left(\frac{\log N}{\log\log N + \log Q}\right)$ worst-case expected step complexity and solo step complexity.*

By specializing the parametrization to focus on particular ranges of interest, we obtain the following two results.

▶ **Corollary 9.** *A Leader Election algorithm with constant solo step complexity is only possible if the number of different values a register can hold is polynomial in N and, hence, only if the number of different processes that can be poised simultaneously poised to access the same register can be polynomial in N.*

▶ **Corollary 10.** *Any (randomized) Leader Election algorithm that has $o\left(\frac{\log N}{\log\log N}\right)$ solo step complexity (with high probability) has value contention that is super-polylogarithmic in N.*

We can also apply our theorem to adaptive Leader Election algorithms, where the step complexity depends on the number of processes that take at least one step, rather than on the total number of processes in the system.

▶ **Corollary 11.** *For value contention $Q = N^{o(1)}$, where $c > 0$ is constant, it is impossible to solve Leader Election with expected solo step complexity (and, hence, with worst-case expected step complexity) $O(f(k))$, where k is the number of participating processes and f is an arbitrary function.*

5 Upper Bounds

We discuss how existing upper bounds relate to our lower bounds. The related algorithms are listed in Table 2.

Table 2 Upper bounds for Leader Election against a strong adversary.

Step Compl.	Total Step Compl.	Q	Reference	Comments
$O(\log N)$	$O(k \log N)$	constant	[1]	with high probability
$O\left(\log_\kappa N\right)$	$O\left(k \log_\kappa N\right)$	κ	[5]	Solves *Weak* Leader Election, κ is the write contention

Leader Election with One-Bit Registers

Afek, Gafni, Tromp and Vitányi presented a Leader Election algorithm for N processes with $O(\log N)$ step complexity with high probability based on a tournament tree [1] using Leader Election for 2 processes at each internal node. Leader Election for 2 processes can be solved by the randomized algorithm of Tromp and Vitányi in expected constant step complexity using 4-valued shared registers [26]. We can then replace each 4-valued register with 4 one-bit registers using the construction of [20]. Thus, the value contention, Q, of the resulting algorithm is 2. The step complexity of this algorithm matches our lower bound to within an $O(\log \log N)$ factor.

Weak Leader Election Based on Splitters

Weak Leader Election differs from Leader Election in that it does not guarantee existence of a winner if there is more than one participating process. Alistarh, Gelashvili and Nadiradze proposed an algorithm for the Weak Leader Election problem with bounded write contention κ [5], which is based on Splitter objects. Their algorithm arranges processes into a complete κ-ary tournament tree, where each node is associated with a Splitter.

Since their tree is κ-ary, the write contention is bounded by κ and, consequently, the value contention is also bounded by κ. The construction in our proof of Theorem 7, ensures that processes do not see other processes, so our lower bound also applies to the Weak Leader Election problem. Therefore, there is an $\Omega\left(\frac{\log N}{\log \log N + \log \kappa}\right)$ lower bound and an almost matching $O\left(\frac{\log N}{\log \kappa}\right)$ upper bound on the step complexity of this problem.

Adaptive RatRace Leader Election

Alistarh, Attiya, Gilbert, Giurgiu, and Guerraoui proposed an algorithm for solving Leader Election, which they call *RatRace* [4], that has $O(\log k)$ step complexity with high probability and $O(N^3)$ space complexity, where k is the number of participating processes.

The algorithm consists of a *primary tree* and a *backup grid*. The primary tree is a complete binary tree of Randomized Splitters of depth $3 \log N$. Every process starts at the root of the primary tree and then either stops or moves to the left or right child depending on the result (`Stop`, `Left` or `Right`) of the corresponding Randomized Splitter. Once a process obtains `Stop`, it tries to move back to the root by participating in an instance of Leader election among 3 processes: the process that received `Stop` from the the Randomized Splitter and one process from each child. The winner of the instance of Leader Election at a node moves

back to its parent, if it has a parent. In the unlikely case when a process reaches a leaf of the tree without obtaining `Stop`, the process accesses the backup grid, which is a $O(N^2)$ grid of deterministic Splitters [4] and instances of Leader Election among 3 processes. The winner of the backup grid and the winner of the instance of Leader Election at the root of the primary tree then participate in a final instance of Leader Election (among 2 processes) to decide the winner of the Leader Election among all N processes. Giakkoupis and Woelfel improved the space complexity of the RatRace algorithm from $O(N^3)$ to $O(N)$ [14] by replacing the backup grid with $\frac{N}{\log N}$ elimination paths of size $4 \log N$ and an elimination path of size N and reducing tree height from $3 \log N$ to $\log N$.

The value contention of a Splitter shared by N processes is N, so the value contention of both these algorithms is N. Corollary 11 proves that adaptive Leader Election is impossible unless Q is polynomial in N. In particular, this means that any adaptive Leader Election algorithm has to use registers of size $\Omega(\log N)$ and it must have configurations in which $N^{\Omega(1)}$ process are poised to write different values to the same register.

6 Discussion

We made a step towards improving Leader Election bounds. Although we did not achieve a matching logarithmic lower bound that depends only on the number of processes, we proved tighter and more general lower bounds than previous work [5] by analyzing a new notion of value contention. Our primary contribution is the establishment of a new lower bound of

$$\Omega\left(\frac{\log N}{\log\log N + \log Q}\right)$$

on the step complexity for any obstruction-free leader election algorithm.

Moreover, we have demonstrated that achieving $o\left(\frac{\log N}{\log\log N}\right)$ step complexity requires value contention to be $\omega(\texttt{polylog}(N))$. On the other hand, we showed that adaptive leader election is impossible unless the value contention is also polynomial in N, i.e. polynomially many processes can be simultaneously poised to write polynomially many different values to the same register. These impossibility results place fundamental limits on the design of efficient leader election algorithms.

Finally, we speculate that our approach stretches the existing approaches to their limit, and that further progress towards a general lower bound will require a more general technique that bounds the information flow achievable in the high-contention scenario where all processors can simultaneously be poised to write different values to the same register.

References

1 Yehuda Afek, Eli Gafni, John Tromp, and Paul M. B. Vitányi. Wait-free test-and-set (extended abstract). In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG)*, pages 85–94, 1992. doi:10.1007/3-540-56188-9_6.

2 Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Stable leader election (extended abstract). In *Proceedings of the 15th International Symposium on Distributed Computing (DISC)*, volume 2180 of *Lecture Notes in Computer Science*, pages 108–122, 2001.

3 Dan Alistarh and James Aspnes. Sub-logarithmic test-and-set against a weak adversary. In *Proceedings of the 25th International Symposium on Distributed Computing (DISC)*, volume 6950 of *Lecture Notes in Computer Science*, pages 97–109, 2011. doi:10.1007/978-3-642-24100-0_7.

4 Dan Alistarh, Hagit Attiya, Seth Gilbert, Andrei Giurgiu, and Rachid Guerraoui. Fast randomized test-and-set and renaming. In *Proceedings of the 24th international Symposium on Distributed Computing (DISC)*, volume 6343 of *Lecture Notes in Computer Science*, pages 94–108, 2010. doi:10.1007/978-3-642-15763-9_9.

5 Dan Alistarh, Rati Gelashvili, and Giorgi Nadiradze. Lower bounds for shared-memory leader election under bounded write contention. In *Proceedings of the 35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 4:1–4:17, 2021. doi:10.4230/LIPICS.DISC.2021.4.

6 Dan Alistarh, Rati Gelashvili, and Giorgi Nadiradze. Lower bounds for shared-memory leader election under bounded write contention, 2022. URL: https://arxiv.org/abs/2108.02802, arXiv:2108.02802.

7 James H. Anderson and Yong-Jik Kim. An improved lower bound for the time complexity of mutual exclusion. *Distributed Computing*, 15(4):221–253, 2002. doi:10.1007/S00446-002-0084-2.

8 Hagit Attiya, Fabian Kuhn, C. Greg Plaxton, Mirjam Wattenhofer, and Roger Wattenhofer. Efficient adaptive collect using randomization. *Distributed Computing*, 18(3):179–188, 2006. doi:10.1007/S00446-005-0143-6.

9 Hagit Attiya and Jennifer Welch. *Distributed Computing. Fundamentals, Simulations, and Advanced Topics (second edition)*. Wiley, 2004.

10 Aryaz Eghbali and Philipp Woelfel. An almost tight RMR lower bound for abortable test-and-set. In *Proceedings of the 32nd International Symposium on Distributed Computing (DISC)*, volume 121 of *LIPIcs*, pages 21:1–21:19, 2018. doi:10.4230/LIPICS.DISC.2018.21.

11 Faith Ellen, Rati Gelashvili, and Leqi Zhu. Revisionist simulations: A new approach to proving space lower bounds. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 61–70, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3212734.3212749.

12 George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. An o(sqrt n) space bound for obstruction-free leader election. In *Proceedings of the 27th International Symposium on Distributed Computing (DISC)*, volume 8205 of *Lecture Notes in Computer Science*, pages 46–60, 2013. doi:10.1007/978-3-642-41527-2_4.

13 George Giakkoupis, Maryam Helmi, Lisa Higham, and Philipp Woelfel. Test-and-set in optimal space. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 615–623, 2015. doi:10.1145/2746539.2746627.

14 George Giakkoupis and Philipp Woelfel. On the time and space complexity of randomized test-and-set. In *Proceedings of the 31st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2012. doi:10.1145/2332432.2332436.

15 George Giakkoupis and Philipp Woelfel. Efficient randomized test-and-set implementations. *Distributed Computing*, 32(6):565–586, 2019. doi:10.1007/S00446-019-00349-Z.

16 Wojciech Golab, Danny Hendler, and Philipp Woelfel. An o(1) RMRs leader election algorithm. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 238–247, 2006. doi:10.1145/1146381.1146417.

17 Maurice Herlihy and Mark Tuttle. Wait-free computation in message-passing systems: Preliminary report. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–362, 1990.

18 Johan Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906.

19 Yong-Jik Kim and James Anderson. A time complexity bound for adaptive mutual exclusion. *Distributed Computing*, 24(6):271–297, 2012.

20 Leslie Lamport. On interprocess communication. part I: Basic formalism. *Distributed Computing*, 1(2):77–85, 1986. doi:10.1007/BF01786227.

21 Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems (TOCS)*, 5(1):1–11, 1987. doi:10.1145/7351.7352.

22 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

23 Mark Moir and James H Anderson. Wait-free algorithms for fast, long-lived renaming. *Science of Computer Programming*, 25(1):1–39, 1995. doi:10.1016/0167-6423(95)00009-H.

24 Gary L. Peterson and Michael J. Fischer. Economical solutions for the critical section problem in a distributed system (extended abstract). In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC)*, pages 91–97, 1977. doi:10.1145/800105.803398.

25 Eugene Styer and Gary L. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 177–191, 1989. doi:10.1145/72981.72993.

26 John Tromp and Paul Vitányi. Randomized two-process wait-free test-and-set. *Distributed Computing*, 15(3):127–135, 2002. doi:10.1007/S004460200071.

27 Paul Turán. On an extremal problem in graph theory. *Matematikai és Fizikai Lapok*, 48:436–452, 1941.

28 Jae-Heon Yang and James H Anderson. Time bounds for mutual exclusion and related problems. In *Proceedings of the 26th Annual ACM symposium on Theory of Computing (STOC)*, pages 224–233, 1994. doi:10.1145/195058.195139.

pod: An Optimal-Latency, Censorship-Free, and Accountable Generalized Consensus Layer

Orestis Alpos
Common Prefix, Athens, Greece

Bernardo David
IT University of Copenhagen (ITU), Denmark
Common Prefix, Athens, Greece

Jakov Mitrovski
Technical University of Munich, Germany
Common Prefix, Athens, Greece

Odysseas Sofikitis
Common Prefix, Athens, Greece
pod network, Athens, Greece

Dionysis Zindros
Common Prefix, Athens, Greece
pod network, Athens, Greece

Abstract

This work addresses the inherent issues of high latency in blockchains and low scalability in traditional consensus protocols. We present pod, a novel notion of consensus whose first priority is to achieve the physically-optimal latency of 2δ, or one round-trip, *i.e.*, requiring only one network trip (duration δ) for writing a transaction and one for reading it.

To accomplish this, we first eliminate inter-replica communication. Instead, clients send transactions directly to all replicas, which independently process transactions and append them to local logs. Replicas assign a timestamp and a sequence number to each transaction in their logs, allowing clients to extract valuable metadata about the transactions and the system state. Later on, clients retrieve these logs and extract transactions (and associated metadata) from them.

Necessarily, this construction achieves weaker properties than a total-order broadcast protocol, due to existing lower bounds. Our work models the primitive of pod and defines its security properties. We then show pod-core, a protocol that satisfies properties such as transaction confirmation within 2δ, censorship resistance against Byzantine replicas, and accountability for safety violations. We show that single-shot auctions can be realized using the pod notion and observe that it is also sufficient for other popular applications.

2012 ACM Subject Classification Security and privacy → Distributed systems security; Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases consensus, censorship resistance, accountability, auctions

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.4

Related Version *Full Version*: https://arxiv.org/abs/2501.14931 [2]

1 Introduction

Despite the widespread adoption of blockchains, a significant challenge remains unresolved: they are inherently slow. The latency from the moment a client submits a transaction to when it is confirmed in another client's view of the blockchain can be prohibitively long for certain applications. Notice that we define latency in terms of the blockchain *liveness* property, referring to finalized, non-reversible outputs: once a transaction is received by a reader, it


© Orestis Alpos, Bernardo David, Jakov Mitrovski, Odysseas Sofikitis, and Dionysis Zindros;
licensed under Creative Commons License CC-BY 4.0



remains in the protocol's output permanently. Moreover, we do not assume "optimistic" or "happy path" scenarios, where transactions might finalize faster under favorable conditions (such as having honest leaders or optimal network conditions).

Indeed, Nakamoto-style blockchain protocols require a large number of rounds in order to achieve consensus on a new block, even when considering the best known bounds [15]. On the other hand, it is known that permissioned protocols for n parties (out of which t are corrupted) realizing traditional notions of broadcast and Byzantine agreement require at least $t+1$ rounds in the synchronous case [1] and at least $2n/(n-t)$ rounds in the asynchronous case [14], even when allowing for digital signatures and probabilistic termination.

In a model where *replicas* maintain the network, *writers* submit transactions, and *readers* read the network, the minimum latency is one network round trip, or 2δ, letting δ denote the actual network delay, as the information must travel from the writers to the replicas and then to the readers. More importantly, we want that any transaction from an honest writer appears in the output of honest readers within 2δ time, regardless of the current value of δ and corrupted parties' actions. In this context, we are motivated by the following question:

Can we realize tasks that blockchains are commonly used for with optimal latency?

We give a positive answer to this question with a protocol realizing pod, a new notion of consensus that trades off traditional agreement properties for optimal latency, while retaining sufficient security guarantees to realize important tasks (*e.g.*, decentralized auctions).

1.1 Our Contributions

In order to motivate the notion of pod, we first introduce the architecture of our protocol, pod-core, which realizes this notion. To achieve the single-round-trip latency, our first key design decision is to eliminate inter-replica communication entirely. Instead, writers send their transactions directly to all replicas. Each replica maintains its own *replica log*, processes incoming transactions independently, and transmits its log to readers on request. Readers then process these replica logs to extract transactions and relevant associated information. See Figure 1 for a summary of the pod-core architecture.

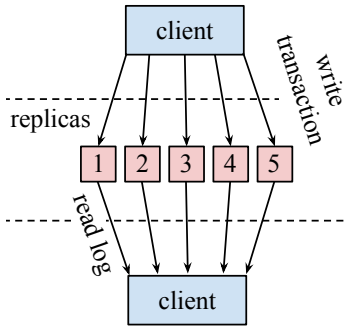


Figure 1 pod-core's simple architecture. A writing client (top) sends a transaction to all replicas (middle). Each replica appends it to its own log and transmits it to the reading client (bottom).

This design raises two important questions. First, what meaningful information can readers derive from replica logs when replicas operate in isolation? Second, given that in two rounds even randomized authenticated broadcast is proven impossible [14], what capabilities can this – necessarily weaker – primitive offer? We demonstrate that, by incorporating simple mechanisms, such as assigning timestamps and sequence numbers to transactions, replicas

can enable readers to extract valuable information beyond mere low-latency guarantees. Furthermore, we show how the properties of pod can enable various applications, including auctions (as shown in Section 6). Specifically, a secure pod delivers the following guarantees (formally defined in Section 3):

- Transaction confirmation within 2δ, with each transaction assigned a *confirmed round*: we say that the transaction becomes *confirmed* at the time indicated by the *confirmed round*.
- Censorship resistance when facing up to β Byzantine and γ omission-faulty replicas, ensuring all confirmed transactions appear in every honest reader's output.
- A *past-perfect round* can be computed by readers, such that the reader is guaranteed to have received all transactions that are or will be confirmed prior to this round, even though not all transactions are strictly ordered.
- Accountability for all safety violations, *i.e.*, if any safety property is violated, at least $\beta + 1$ replicas can be identified as misbehaving.

In particular, our Protocol pod-core, presented in Section 4, realizes the notion of pod with the properties above, supporting a continuum of two adversarial models: up to β Byzantine replicas and up to γ omission-faulty replicas, out of a total of $n > 5\beta + 3\gamma$ replicas. Protocol pod-core requires no expensive cryptographic primitives or setup beyond digital signatures and a PKI registering replicas' public keys. We showcase pod-core's efficiency by means of experiments with a prototype implementation presented in Section 5. Our experiments show that even with 1000 replicas distributed around the world, the latency achieved by our protocol is just under double (resp. about 5 times) the round-trip time between writer and reader clients with security against omission-faulty (resp. Byzantine) replicas.

1.2 Technical Overview

We consider that time proceeds in *rounds*, and that parties (replicas and clients) know the current round, so we can express *timestamps* in terms of rounds. The output of pod associates each transaction tx with timestamp values $\mathsf{r}_{\min} \geq 0$ (minimum round), $\mathsf{r}_{\max} \leq \infty$ (maximum round) and r_{conf} (*confirmed round*). We call these values the *trace* of tx, and they evolve over time. Initially we have $\mathsf{r}_{\text{conf}} = \perp$ but later we get $\mathsf{r}_{\text{conf}} \neq \perp$, when a transaction is *confirmed*. The protocol guarantees *confirmation within u rounds*, meaning that, at most u rounds after tx was written, every party who reads the pod will see tx as confirmed with some $\mathsf{r}_{\text{conf}} \neq \perp$. The protocol also guarantees that $\mathsf{r}_{\min} \leq \mathsf{r}_{\text{conf}} \leq \mathsf{r}_{\max}$, a property we call *confirmation bounds*: while each party reads different values $\mathsf{r}_{\min}, \mathsf{r}_{\max}, \mathsf{r}_{\text{conf}}$ for the same tx, pod guarantees that values read by different parties stay within these limits.

When clients *read* the pod, they obtain a pod data structure $D = (\mathsf{T}, \mathsf{r}_{\text{perf}})$, where T is the set of transactions and their traces and r_{perf} is a *past-perfect* round. The *past-perfection* safety property guarantees that T contains *all* transactions that every other honest party will ever read with a confirmed round smaller than r_{perf}. A pod also guarantees *past-perfection within w*, meaning that r_{perf} is at most w rounds in the past.

In summary, pod provides *past-perfection* and *confirmation bounds* as safety properties, ensuring parties cannot be blindsided by transactions suddenly appearing as confirmed too far in the past, and that the different (and continuously changing) transaction timestamps stay in a certain range. The liveness properties of *confirmation within u* and *past-perfection within w* ensure that new transactions get confirmed within a bounded delay, and that each party's past-perfect round must be constantly progressing.

Besides introducing the notion of pod, we present protocol pod-core, which realizes this notion while requiring minimal interaction among parties and achieving optimal latency, *i.e.*, optimal parameters $u = 2\delta$ and $w = \delta$, where δ is the current network delay (not a delay upper bound, which we assume to be unknown). The only communication is between each client and the replicas. Writing a transaction tx to pod-core only requires clients to send tx to the replicas, who each assign a timestamp ts (their current time) and a *sequence number* sn to tx and return a signature on $(\mathsf{tx}, \mathsf{ts}, \mathsf{sn})$. When reading the pod, the client simply requests each replica's log of transactions, validates the responses, and determines $\mathsf{r}_{\min}$, $\mathsf{r}_{\max}$, and r_{conf} from the received timestamps. Protocol pod-core supports a continuum of mixed adversarial models, tolerating up to β Byzantine *and* at the same time up to γ additional omission-faulty replicas.

1.3 Related work

Many previous works have lowered the latency of ordering transactions. HotStuff [27] uses three rounds of all-to-leader and leader-to-all communication pattern, which results in a latency (measuring from the moment a client submits a transaction until in appears in the output of honest replicas) of 8δ in the happy path. Jolteon [16], Ditto [16], and HotStuff-2 [19] are two-round versions of HotStuff with end-to-end latency of 5δ. MoonShot [10] allows leaders to send a new proposal every δ time, before receiving enough votes for the previous one, but still achieves an end-to-end latency of 5δ. In the "DAG-based" line of word, Tusk [8] achieves and end-to-end latency of 7δ, the partially-synchronous version of BullShark [23] an end-to-end latency of 5δ, and Mysticeti [3] an end-to-end latency of 4δ. All these protocols aim at total-order properties and have their lower latency is inherently restricted by lower bounds, whereas pod starts from the single-round-trip latency requirement and explores the properties that can be achieved.

The redundancy of consensus for implementing payment systems has been recognized by previous works [18, 22, 7, 5]. The insight is that total transaction order is not required in the case that each account is controlled by one client. Instead, a partial order is sufficient, ensuring that, if transactions tx_1 and tx_2 are created by the same client, then every party outputs them in the same order. This requirement was first formalized by Guerraoui *et al.* [18] as the *source-order property*. The constructions of Guerraoui *et al.* [18] and FastPay [5] require clients to maintain sequence numbers. ABC [22] requires clients to reference all previous transaction in a DAG (including its own last transaction). Cheating clients might lose liveness [5, 18, 22], but equivocating is not possible. To the best of our knowledge, previous work in the consensusless literature has not considered or achieved a property like our *past-perfection*, which we show sufficient for implementing auctions.

2 Preliminaries

Notation. We denote by $\mathbb{N}$ the set of natural numbers including 0. Letting L be a bounded sequence, we denote by $L[i]$ the i^{th} element (starting from 0), and by $|L|$ its length. Negative indices address elements from the end, so $L[-i]$ is the i^{th} element from the end, and $L[-1]$ in particular is the last. The notation $L[i{:}]$ means the subarray of L from i onwards, while $L[{:}j]$ means the subsequence of L up to (but not including) j. We denote an empty sequence by $[\,]$. We denote the concatenation of sequences L_1 and L_2 by $L_1 \,\|\, L_2$.

Pseudocode notation. Command "**require** P" causes a function to terminate immediately and return false if P evaluates to false. Notation "**upon** e" causes a block of code to be executed when event e occurs. Notations "$\langle MSG \rangle \leftarrow p$" and "$\langle MSG \rangle \rightarrow p$" denote receiving

and sending a message MSG from and to party p, respectively. Finally, $x : a \in A \to b \in B$ denotes that variable x is a map from elements of type A to elements of type B. When obvious from the context, we do not explicitly write the types A or B. For a map x, the operations $x.keys()$ and $x.values()$ return all keys and all values in x, respectively. With $\emptyset$ we denote an empty map.

Parties. We consider n *replicas* $R = \{R_1, \ldots, R_n\}$ and an unknown number of *clients*. Parties are *stateful*, *i.e.*, store *state* between executions of different algorithms. We assume that replicas are known to all parties and register their public keys (for which they have corresponding secret keys) in a Public Key Infrastructure (PKI). Clients do not register keys in the PKI.

Adversarial model. We call a party (replica or client) *honest*, if it follows the protocol, and *malicious* otherwise. We assume *static corruptions*, *i.e.*, the set of malicious replicas is decided before the execution starts and remains constant. This work uses a combination of two adversarial models, the *Byzantine* and the *omission* models. In the *Byzantine* model, corrupted replicas are malicious and may deviate arbitrarily from the protocol. The adversary has access to the internal state and secret keys of all corrupted parties. We denote by $\beta \in \mathbb{N}$ the number of Byzantine replicas in an execution. The Byzantine adversary is modelled as a probabilistic polynomial time overarching entity that is invoked in the stead of every corrupted party. That is, whenever the turn of a corrupted party comes to be invoked by the environment, the adversary is invoked instead. In the *omission* model, corrupted replicas may only deviate from the protocol by dropping messages that they were supposed to send, but follow the protocol otherwise. Observe that this includes crash faults, where replicas crash (*i.e.* stop execution) and remain crashed until the end of the execution of an algorithm. We denote by $\gamma \in \mathbb{N}$ the number of omission-faulty replicas in an execution.

Modeling time. We assume that time proceeds in discrete *rounds*, and parties have clocks allowing them to determine the current round. For simplicity, our analysis will assume *synchronized clocks*. Notice that although we assume synchronized clocks as a setup, clock synchronization can be achieved in partially synchronous networks [11] using existing techniques [21], also in the case where replicas gradually join the network [26]. By *timestamp* we refer to a round number assigned to some event.

Modeling network. We denote by $\delta \in \mathbb{N}$ the actual delay (measured in number of rounds) it takes to deliver a message between two honest parties, a number which is *finite* but *unknown* to all parties. We denote by $\Delta \in \mathbb{N}$ an upper bound on this delay, *i.e.*, $\delta \leq \Delta$, which is also *finite*. In the *synchronous* model, Δ is *known* to all parties. In the *partially synchronous* model [11], Δ is *unknown* but still finite, *i.e.*, all messages are eventually delivered. A protocol is called *responsive* if it does not rely on knowledge of Δ and its liveness guarantees depend only on the actual network delay δ.

Digital signatures

We assume that replicas (and auctioneers in bidset-core) authenticate their messages with digital signatures. A digital signature scheme consists of the following three algorithms, satisfying the EUF-CMA security [17]: (1) $KeyGen(1^\kappa)$: The key generation algorithm takes as input a security parameter κ and outputs a secret key sk and a public key pk. (2) $Sign(\mathsf{sk}, m) \to \sigma$: The signing algorithm takes as input a private key sk and a message

$m \in \{0,1\}^*$ and returns a signature σ. (3) *Verify*$(\mathsf{pk}, m, \sigma) \to b \in \{0,1\}$: The verification algorithm takes as input a public key pk, a message m, and a signature σ, and outputs a bit $b \in \{0,1\}$. We say σ is a *valid* signature on m with respect to pk if *Verify*$(pk, m, \sigma) = 1$.

Accountable safety

Taking a similar approach as Neu, Tas, and Tse [20, Def. 4], we define *accountable safety* through an *identification function*.

▶ **Definition 1** (Transcript and partial transcript). *We define as* transcript *the set of all network messages sent by all parties in an execution of a protocol. A* partial transcript *is a subset of a transcript.*

▶ **Definition 2** (β-Accountable safety). *A protocol satisfies* accountable safety *with* resilience β *if its interface contains a function* $\mathsf{identify}(T) \to \tilde{R}$*, which takes as input a partial transcript* T *and outputs a set of replicas* $\tilde{R} \subset R$*, such that the following conditions hold except with negligible probability.*

Correctness: *If safety is violated, then there exists a partial transcript* T*, such that* $\mathsf{identify}(T) \to \tilde{R}$ *and* $|\tilde{R}| > \beta$*.*

No-framing: *For any partial transcript* T *produced during an execution of the protocol, the output of* $\mathsf{identify}(T)$ *does not contain honest replicas.*

▶ Remark 3. For simplicity, we have defined the transcript based on messages sent by *all* replicas. We can also define a *local transcript* as the set of messages observed by a single party. As will become evident from the implementation of $\mathsf{identify}()$, in practice, adversarial behavior can be identified from the local transcripts of a single party or of a pair of parties.

3 Modeling pod

In this section, we introduce the notion of a pod, a distributed protocol where clients can *read* and *write* transactions. We first define the basic data structures of a pod protocol.

▶ **Definition 4** (Transaction trace and trace set). *The* transaction trace *of a transaction* $\mathsf{tx} \in \{0,1\}^*$ *is a tuple containing the values* $(\mathsf{tx}, r_{min}, r_{max}, r_{conf})$*, which change during the execution of a* pod *protocol. We call* $r_{min} \in \mathbb{N}$ *the* minimum round*,* $r_{max} \in \mathbb{N} \cup \{\infty\}$ *the* maximum round*,* $r_{conf} \in \mathbb{N} \cup \{\perp\}$ *the* confirmed round*. We denote by* $r_{max} = \infty$ *an unbounded maximum round and by* $r_{conf} = \perp$ *an undefined confirmed round. We also denote these values as* $\mathsf{tx}.r_{min}, \mathsf{tx}.r_{max}$*, and* $\mathsf{tx}.r_{conf}$*. A* trace set T *is a set of transaction traces* $\{(\mathsf{tx}, r_{min}, r_{max}, r_{conf}) \mid \mathsf{tx} \in \{0,1\}^*\}$.

▶ **Definition 5** (Confirmed transaction). *A transaction with confirmed round* r_{conf} *is called* confirmed *if* $r_{conf} \neq \perp$*, and* unconfirmed *otherwise.*

▶ **Definition 6** (Pod data structure). *A* pod data structure D *is a tuple* (T, r_{perf})*, where* T *is a trace set and* r_{perf} *is a round number called the* past-perfect round*.*

We denote the components of a pod data structure as $D.\mathsf{T}$ and $D.\mathsf{r}_{\text{perf}}$. We write $\mathsf{tx} \in D.\mathsf{T}$ if an entry $(\mathsf{tx}, \cdot, \cdot, \cdot)$ exists in $D.\mathsf{T}$. We remark that transactions in T may be confirmed on unconfirmed. Moreover, r_{perf} will be used to define a completeness property on T (the *past-perfection* property of pod).

▶ **Definition 7** (Auxiliary data). *We associate with a pod data structure D some auxiliary data C, which will be used to validate D. The exact implementation of C is irrelevant for the definition of pod; however, it is helpful to mention that in pod-core it will be a tuple $C = (C_{pp}, \mathbb{C}_{tx})$, where C_{pp} will be a* past-perfection certificate *and $\mathbb{C}_{tx}$ a map from each transaction tx in $D.\mathsf{T}$ to a* transaction certificate *C_{tx} for tx. Both contain digital signatures.*

▶ **Definition 8** (Interface of a pod). *A pod protocol has the following interface.*

- *write(tx): It writes a transaction tx to the pod.*
- *read() $\to (D, C)$: It outputs a pod data structure $D = (\mathsf{T}, r_{perf})$ and auxiliary data C.*

We say that a client *reads the* pod when it calls *read*(). If tx appears in T, we say that the client *observes* tx and, if $\mathsf{tx}.\mathsf{r}_{\text{conf}} \neq \perp$, we say that the client *observes tx as confirmed.*

▶ **Definition 9** (Validity function). *Apart from its interface functions, a pod protocol also specifies a computable, deterministic, and non-interactive function valid(D, C) that takes as input a pod data structure D and auxiliary data C and outputs a boolean value. We say that a pod data structure D is* valid *if valid$(D, C) = \mathsf{true}$.*

▶ **Definition 10** (View of the pod). *We call* view of the *pod and denote by $D^{\mathsf{c}}_{\mathsf{r}}$ the data structure returned by read(), where read() is invoked by client c and the output is produced at round r. We remark that r denotes the round when read() outputs, as the client may have invoked it at an earlier round.*

We now introduce the basic definition of a *secure* pod protocol, as well as some additional properties (*timeliness* and *monotonicity*) that it may satisfy.

▶ **Definition 11** (Secure pod). *A protocol is a* secure *pod if it implements the pod interface of Definition 8 and specifies a validity function valid(), such that the following properties hold.*

(Liveness) Completeness: *Honest clients always output a valid pod data structure. That is, if read() returns (D, C) to an honest client, then valid$(D, C) = \mathsf{true}$.*

(Liveness) Confirmation within u: *Transactions of honest clients become confirmed after at most u rounds. Formally, if an honest client c writes a transaction tx at round r, then for any honest client c' (including $\mathsf{c} = \mathsf{c}'$) it holds that $\mathsf{tx} \in D^{\mathsf{c}'}_{r+u}$ and $\mathsf{tx}.r_{conf} \neq \perp$.*

(Liveness) Past-perfection within w: *Rounds become past-perfect after at most w rounds. Formally, for any honest client c and round $\mathsf{r} \geq w$, it holds that $D^{\mathsf{c}}_{\mathsf{r}}.r_{perf} \geq \mathsf{r} - w$.*

(Safety) Past-perfection: *A valid pod D contains all transactions that may ever obtain a confirmed round smaller than $D.r_{perf}$. Formally, the adversary cannot output (D_1, C_1) and (D_2, C_2) to the network, such that valid$(D_1, C_1) \wedge$ valid(D_2, C_2) and there exists a transaction tx such that $(\mathsf{tx}, r^1_{min}, r^1_{max}, r^1_{conf}) \notin D_1.\mathsf{T}$ and $(\mathsf{tx}, r^2_{min}, r^2_{max}, r^2_{conf}) \in D_2.\mathsf{T}$ and $r^2_{conf} \neq \perp$ and $r^2_{conf} < D_1.r_{perf}$.*

(Safety) Confirmation bounds: *The values r_{min} and r_{max} bound the confirmed round that a transaction may ever obtain. Formally, the adversary cannot output (D_1, C_1) and (D_2, C_2) to the network, such that valid$(D_1, C_1) \wedge$ valid(D_2, C_2) and there exists a transaction tx such that $(\mathsf{tx}, r^1_{min}, r^1_{max}, r^1_{conf}) \in D_1.\mathsf{T}$ and $(\mathsf{tx}, r^2_{min}, r^2_{max}, r^2_{conf}) \in D_2.\mathsf{T}$ and $r^1_{min} > r^2_{conf}$ or $r^1_{max} < r^2_{conf}$.*

The *confirmation bounds* property gives $\mathsf{r}^1_{\text{min}} \leq \mathsf{r}^2_{\text{conf}} \leq \mathsf{r}^1_{\text{max}}$, for $\mathsf{r}^1_{\text{min}}, \mathsf{r}^1_{\text{max}}, \mathsf{r}^2_{\text{conf}}$ computed by honest clients, but it does not guarantee anything about the values of $\mathsf{r}^1_{\text{min}}$ and $\mathsf{r}^1_{\text{max}}$ (for example, it could trivially be $\mathsf{r}^1_{\text{min}} = 0$ and $\mathsf{r}^1_{\text{max}} = \infty$). To this purpose we define an additional property of pod, called *timeliness*. Previous work has observed a similar property as orthogonal to safety and liveness [25].

▶ **Definition 12** (pod θ-timeliness for honest transactions). *A pod protocol is* θ-timely *if it is a secure pod, as per Definition 11, and for any honest clients* $\mathsf{c}_1, \mathsf{c}_2$*, if* c_1 *writes transaction* tx *in round* r *and* c_2 *has view* $D_{r'}^{\mathsf{c}_2}$ *in round* r'*, such that* $(\mathsf{tx}, r_{min}, r_{max}, r_{conf}) \in D_{r'}^{\mathsf{c}_2}.T$*, then: (1)* $r_{conf} \in (r, r+\theta]$*; (2)* $r_{max} \in (r, r+\theta]$*; (3)* $r_{max} - r_{min} < \theta$*, implying* $r_{min} \neq 0$ *and* $r_{max} \neq \infty$.

Moreover, a pod protocol allows $\mathsf{r}_{\min}$, $\mathsf{r}_{\max}$, r_{conf} to change during an execution – for example, clients in construction pod-core will update them when they receive votes from replicas. In the full version of this paper [2, Appendix A] we define *monotonicity* properties that impose restrictions on how these values evolve.

We conclude this section with some visual examples in Figures 2 and 3.

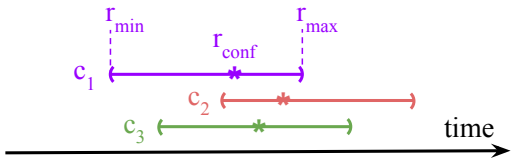


Figure 2 The same transaction in the view of three different pod clients. Each client assigns it a minimum round $\mathsf{r}_{\min}$ and a maximum round $\mathsf{r}_{\max}$. If it gets confirmed, the confirmation round r_{conf} will be between these two values. The r_{conf} that each client locally computes respects the bounds of each other client.

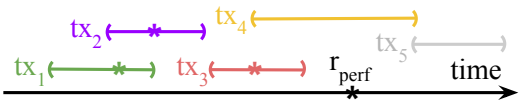


Figure 3 A possible view of a single pod client. Transactions $\mathsf{tx}_1, \mathsf{tx}_2, \mathsf{tx}_3$ are confirmed, tx_4 is not yet confirmed. A client also derives a past-perfect round r_{perf}. No transaction other than $\mathsf{tx}_1, \mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_4$ may obtain $\mathsf{r}_{\text{conf}} \leq \mathsf{r}_{\text{perf}}$. There may exist tx_5 for which the client has not received votes, but tx_5 cannot obtain $\mathsf{r}_{\text{conf}} \leq \mathsf{r}_{\text{perf}}$.

4 Protocol pod-core

Before we present protocol pod-core, we define basic concepts and structures.

▶ **Definition 13** (Vote). *A* vote *is a tuple* $\mathsf{vote} = (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R)$*, where* tx *is a transaction,* ts *is a timestamp,* sn *is a sequence number,* σ *is a signature, and* R *is a replica. A vote is* valid *if* σ *is a valid signature on message* $m = (\mathsf{tx}, \mathsf{ts}, \mathsf{sn})$ *with respect to the public key* pk_R *of replica* R.

▶ Remark 14 (Sequence numbers, session identifiers, streaming algorithm). Honest clients process votes from each replica in the same order, namely in order of increasing timestamps. For this, a replica maintains a *sequence number* which it increments and includes every time it assigns a timestamp to a transaction. We also assume that all messages between clients and replicas are concatenated with a *session identifier (sid)*, which is unique for each concurrent execution of the protocol and included in all messages signed by the replicas. Finally, the client protocol we show in Protocol 1 is *streaming*, that is, clients maintain a connection to the replicas, and *stateful*, that is, they persist their state (received transactions and votes) across all invocations of *write*() and *read*().

Past-perfection and transaction certificates. In pod-core, clients store certain votes which they output upon *read*() as part of the *certificate* C, which will be used to prove the validity of the returned D and for accountability in case of safety violations. Specifically, C consists of two parts, $C = (C_{\text{pp}}, \mathbb{C}_{\text{tx}})$: the *past-perfection certificate* C_{pp} contains, for each replica, the vote on the most recent timestamp received from that replica. It is implemented as a map from replicas to votes, i.e., $C_{\text{pp}} : R \to \mathsf{vote}$. The *transaction certificate* $\mathbb{C}_{\text{tx}}$ contains, for

each transaction, all valid votes received for it. It is implemented as a map from transactions to a map from replicas to votes, i.e., $\mathbb{C}_{\text{tx}} : \mathsf{tx} \to C_{\text{tx}}$ and $C_{\text{tx}} : R \to \mathsf{vote}$. We remark that C_{pp} can be derived by taking the union of certificates C_{tx} for all transactions and keeping the most recent vote for each replica, but we define C_{pp} explicitly for clarity and readability.

▶ **Protocol 1** (pod-core). *Protocol pod-core is executed by n replicas that follow the steps of Algorithm 1 and an unknown number of clients that follow the steps of Algorithms 2 and 3 with parameters β, γ and α, where β denotes the number of Byzantine replicas and γ the number of omission-faulty replicas (in addition to the Byzantine) and $\alpha = n - \beta - \gamma$ is the number of honest replicas.*

4.1 Replica code

Algorithm 1 Protocol pod-core: Code for a replica R_i, where sk denotes its secret signing key.

```
1:  C                                              ▷ The set of all connected clients
2:  nextsn                                         ▷ The next sequence number to assign to votes
3:  replicaLog                                     ▷ The transaction log or the replica

4:  upon init() do                                 ▷ Called once when the replica is initialized
5:      C ← ∅; nextsn ← 0; replicaLog ← [ ]
6:  end upon

7:  upon ⟨CONNECT ⟩ ← c do                         ▷ Called when a new client c connects to the replica
8:      C ← C ∪ {c}
9:      for (tx, ts, sn, σ) ∈ replicaLog do
10:         ⟨VOTE (tx, ts, sn, σ, R_i)⟩ → c
11:     end for
12: end upon

13: upon ⟨WRITE tx⟩ ← c do                         ▷ Called when a client c writes a transaction tx
14:     if replicaLog[tx] ≠ ⊥ then return          ▷ Ignore duplicate transactions
15:     doVote(tx)
16: end upon

17: function doVote(tx)
18:     ts ← round(); sn ← nextsn; σ ← Sign(sk, (tx, ts, sn))
19:     replicaLog ← replicaLog ‖ (tx, ts, sn, σ)
20:     for c ∈ C do ⟨VOTE (tx, ts, sn, σ, R_i)⟩ → c
21:     nextsn ← nextsn + 1
22: end function

23: upon end round do                              ▷ Executed at the end of each round
24:     tx ← HEARTBEAT‖round()
25:     doVote(tx)
26: end upon
```

We show the pseudocode for the replica code in Algorithm 1. The state of a replica (lines 1–3) contains replicaLog, a log implemented as a sequence of votes $(\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_i)$ created by the replica, where ts is the timestamp assigned by the replica to tx, sn is a sequence number, and σ its signature. When the replica receives $\langle CONNECT \rangle$ from a client c, it appends c to its set of connected clients and sends to c all entries in replicaLog (lines 7–12).

When it receives $\langle \mathit{WRITE}\ \mathsf{tx}\rangle$, a replica first checks whether it has already seen tx, in which case the message is ignored. Otherwise, it assigns tx a timestamp ts equal its local round number and the next available sequence number sn, and signs the message $(\mathsf{tx}, \mathsf{ts}, \mathsf{sn})$ (line 18). Honest replicas use incremental sequence numbers for each transaction, implying that a vote with a larger sequence number than a second vote will have a larger or equal timestamp than the second. The replica appends $(\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma)$ to $\mathsf{replicaLog}$, and sends it via a $\langle \mathit{VOTE}\ (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_i)\rangle$ message to all connected clients (line 20).

Heartbeat messages. Clients maintain a *most-recent timestamp* variable $\mathsf{mrt}[R_j]$ for each replica. This is updated every time they receive a vote and is crucial for computing the past-perfect round r_{perf}. To make sure that clients update $\mathsf{mrt}[R_j]$ even when R_j does not have any new transactions in a round, we have replicas send a vote on a dummy HEARTBEAT transaction the end of each round (lines 23–26). An obvious practical optimization is to send HEARTBEAT only for rounds when no other transactions were sent. When received by a client, a HEARTBEAT is handled as a vote (i.e., it triggers line 11). To avoid being considered a duplicate vote by clients (see line 33 in Algorithm 2), replicas append the round number to the HEARTBEAT transaction.

4.2 Client code

The state of a client is shown in Algorithm 2 in lines 1–6. Variable tsps is a map from transactions tx to a map from replicas R to timestamps ts. The state gets initialized in lines 7–10. At initialization the client also sends a $\langle \mathit{CONNECT}\ \rangle$ message to each replica, which initiates a streaming connection from the replica to the client.

Receiving votes. A client maintains a connection to each replica and receives votes through $\langle \mathit{VOTE}\ (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)\rangle$ messages (lines 11–16). When a vote is received from replica R_j, the client first verifies the signature σ under R_j's public key (line 30). If invalid, the vote is ignored. Then the client verifies that the vote contains the next sequence number it expects to receive from replica R_j (line 31). If this is not the case, the vote is *backlogged* and given again to the client at a later point (the backlogging functionality is not shown in the pseudocode). The client then checks the vote against previous votes received from R_j. First, ts must be greater or equal to $\mathsf{mrt}[R_j]$, the most recent timestamp returned by replica R_j. Second, the replica must have not previously sent a different timestamp for tx. If both checks pass, the client updates $\mathsf{mrt}[R_j]$ and $\mathsf{tsps}[\mathsf{tx}][R_j]$ with ts (line 34). The client also updates C_{pp} and $\mathbb{C}_{\text{tx}}$ for each valid vote (lines 13–14).

If any of these checks fail, the client ignores the vote, since both of these cases constitute *accountable* faults: In the first case, the client can use the message $\langle \mathit{VOTE}\ (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)\rangle$ and the vote it received when it updated $\mathsf{mrt}[R_j]$ to prove that R_j has misbehaved. In the second case, it can use $\langle \mathit{VOTE}\ (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)\rangle$ and the previous vote it has received for tx. The $\mathsf{identify}()$ function we show in Algorithm 8 can detect such misbehavior

Writing to and reading from pod. Clients interact with a pod using the *write*(tx) and *read*() functions. In order to write a transaction tx, a client sends $\langle \mathit{WRITE}\ \mathsf{tx}\rangle$ to each replica (lines 17–19). Since the construction is stateful and streaming, the client state contains at all times the latest view the client has of the pod. Hence, *read*() operates on the local state (lines 20–24). It returns all the transactions the client has received so far and their traces, and the current past-perfect round r_{perf}. We will show the details of *computeTxSet*() in Algorithm 3. As per the pod interface, *read*() also returns auxiliary data C, which has two parts: the past-perfection certificate C_{pp} and a list of transaction certificates C_{tx} (line 23). Note that tsps.*keys()* on Algorithm 3 returns all entries in tsps.

Algorithm 2 Protocol pod-core: Code for a client, part 1.

1: $\mathcal{R} = \{R_1, \ldots, R_n\}$; $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ ▷ All replicas and their public keys
2: $\mathsf{mrt} : R \to \mathsf{ts}$ ▷ The most recent timestamp returned by each replica
3: $\mathsf{nextsn} : R \to \mathsf{sn}$ ▷ The next sequence number expected by each replica
4: $\mathsf{tsps} : \mathsf{tx} \to (R \to \mathsf{ts})$ ▷ Timestamp received for each tx from each replica
5: $C_{\text{pp}} : R \to \mathsf{vote}$ ▷ Past-perfection certificate: most recent vote from each replica
6: $\mathbb{C}_{\text{tx}} : \mathsf{tx} \to C_{\text{tx}}$, where $C_{\text{tx}} : R \to \mathsf{vote}$ ▷ Transaction certificates

7: **upon** *init*() **do** ▷ Called once when the client is initialized
8: *initState*()
9: **for** $R_j \in \mathcal{R}$ **do:** $\langle CONNECT \rangle \to R_j$
10: **end upon**

11: **upon** $\langle VOTE\ (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)\rangle \leftarrow R_j$ **do** ▷ Called when client receives a vote
12: **if** *processVote*$(\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)$ **then**
13: $C_{\text{pp}}[R_j] \leftarrow (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)$ ▷ Keep most recent vote from R_j in C_{pp}
14: $\mathbb{C}_{\text{tx}}[\mathsf{tx}][R_j] \leftarrow (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)$ ▷ Keep all votes for tx in C_{tx}
15: **end if**
16: **end upon**

17: **function** *write*(tx) ▷ Part of pod interface, used to write a new transaction
18: **for** $R_j \in \mathcal{R}$ **do:** $\langle WRITE\ \mathsf{tx}\rangle \to R_j$
19: **end function**

20: **function** *read*() ▷ Part of pod interface, used to read all transactions
21: $\mathsf{T} \leftarrow$ *computeTxSet*$(\mathsf{tsps}, \mathsf{mrt})$ ▷ Shown in Algorithm 3
22: $\mathsf{r}_{\text{perf}} \leftarrow$ *computePastPerfectRound*(mrt) ▷ Shown in Algorithm 3
23: $D \leftarrow (\mathsf{T}, \mathsf{r}_{\text{perf}})$; $C \leftarrow (C_{\text{pp}}, \mathbb{C}_{\text{tx}})$; **return**(D, C)
24: **end function**

25: **function** *initState*()
26: $\mathsf{tsps} \leftarrow \emptyset$; $\mathbb{C}_{\text{tx}} \leftarrow \emptyset$
27: **for** $R_j \in \mathcal{R}$ **do:** $\mathsf{mrt}[R_j] \leftarrow 0$; $C_{\text{pp}}[R_j] \leftarrow \perp$; $\mathsf{nextsn}[R_j] = -1$
28: **end function**

29: **function** *processVote*$(\mathsf{tx}, \mathsf{ts}, \mathsf{sn}, \sigma, R_j)$ ▷ Validate vote and update local state
30: **require** *Verify*$(\mathsf{pk}_j, (\mathsf{tx}, \mathsf{ts}, \mathsf{sn}), \sigma)$ ▷ Otherwise, vote is invalid
31: **require** $\mathsf{sn} = \mathsf{nextsn}[R_j]$ ▷ Otherwise, vote cannot be processed yet
32: **require** $\mathsf{ts} \geq \mathsf{mrt}[R_j]$ ▷ Otherwise, R_j has sent old timestamp
33: **require** $\mathsf{tsps}[\mathsf{tx}][R_j] = \perp$ **or** $\mathsf{tsps}[\mathsf{tx}][R_j] = \mathsf{ts}$ ▷ Otherwise, vote is duplicate
34: $\mathsf{nextsn}[R_j] \leftarrow \mathsf{nextsn}[R_j] + 1$; $\mathsf{mrt}[R_j] \leftarrow \mathsf{ts}$; $\mathsf{tsps}[\mathsf{tx}][R_j] \leftarrow \mathsf{ts}$
35: **end function**

Computing the trace values and the past-perfect round. Function *computeTxSet*() (Algorithm 3), computes the current transaction set from the timestamps received so far. A transaction becomes confirmed when the client receives α votes for it, in which case r_{conf} is the median of all received timestamps (lines 6–8). The computation of $\mathsf{r}_{\text{min}}, \mathsf{r}_{\text{max}}$, and r_{perf} is done using the functions *minPossibleTs*(), *maxPossibleTs*(), and *computePastPerfectRound*(), respectively. Function *minPossibleTs*() gets the timestamps timestamps from each replica on tx and the most recent timestamps mrt. It fills a missing timestamp from replica R_j with

Algorithm 3 Protocol pod-core: Client code, part 2. Functions to compute trace values and past-perfect round. The code is parametrized with β, the number of Byzantine replicas expected by the client, and γ, the number of omission-faulty replicas, and $\alpha = n - \beta - \gamma$ for n replicas.

```
1:  function computeTxSet(tsps, mrt)
2:      T ← ∅
3:      for tx ∈ tsps.keys() do                          ▷ loop over all received transactions
4:          r_min ← minPossibleTs(tsps[tx], mrt) ; r_max ← maxPossibleTs(tsps[tx])
5:          r_conf ← ⊥ ; timestamps ← [ ]
6:          if |tsps[tx].keys()| ≥ α then
7:              for R_j ∈ tsps[tx].keys() do: timestamps ← timestamps ‖ tsps[tx][R_j]
8:              r_conf ← median(timestamps)
9:          end if
10:         T ← T ∪ {(tx, r_min, r_max, r_conf)}
11:     end for
12:     return T
13: end function

14: function minPossibleTs(timestamps, mrt)
15:     for R_j ∈ R do
16:         if timestamps[R_j] = ⊥ then timestamps ← timestamps ‖ [mrt[R_j]]
17:     end for
18:     sort timestamps in increasing order of timestamps
19:     timestamps ← [0, β times ..., 0] ‖ timestamps          ▷ omitted altogether if β = 0
20:     return median(timestamps[: α])
21: end function

22: function maxPossibleTs(timestamps)
23:     for R_j ∈ R do
24:         if timestamps[R_j] = ⊥ then timestamps ← timestamps ‖ [∞]
25:     end for
26:     sort timestamps in increasing order of timestamps
27:     timestamps ← timestamps ‖ [∞, β times ..., ∞]          ▷ omitted altogether if β = 0
28:     return median(timestamps[−α :])
29: end function

30: function computePastPerfectRound(mrt)
31:     sort mrt in increasing order
32:     mrt ← [0, β times ..., 0] ‖ mrt                        ▷ omitted altogether if β = 0
33:     return median(mrt[: α])
34: end function

35: function median(Y) : return Y[⌊ |Y|/2 ⌋]
```

$\mathsf{mrt}[R_j]$ (line 16), the minimum timestamp that can ever be accepted from R_j (see the check in line 32 of Algorithm 2). It then prepends β times the 0 value, pessimistically assuming that up to β replicas will try to bias tx by sending a timestamp 0 to other clients. It then returns the median of the α smallest timestamps, which, again pessimistically, are the smallest timestamps another client may use to confirm tx. Function *maxPossibleTs*() is analogous, filling a missing vote with ∞ (line 24) and appending the ∞ value, the worst-case timestamp

that Byzantine replicas may send to other clients. Finally, *computePastPerfectRound*() is similar to *minPossibleTs*() but it operates on timestamps mrt. As honest clients will not accept a timestamp smaller than mrt on any future transaction (line 32 of Algorithm 2), the returned value bounds from below the confirmed round of *any* transaction *not yet seen*.

4.3 Validation function

The validation function *valid*() allows a pod client to verify that a given pod data structure D satisfies the security properties of pod (Definition 11) *without* necessarily communicating with pod replicas. The function *valid*() for pod-core is shown in Section A. The verifier repeats the logic of an honest client: it is initialized in the same way as a pod client, goes through the votes found in $\mathbb{C}_{\text{tx}}$, and checks that the resulting values match the ones in D.

4.4 Analysis

▶ **Theorem 15** (pod-core security). *Assume that the network is partially synchronous with actual network delay δ, that β is the number of Byzantine replicas, γ the number of omission-faulty replicas, $\alpha = n - \beta - \gamma$ the confirmation threshold, and $n \geq 5\beta + 3\gamma + 1$ the total number of replicas. Protocol pod-core (Protocol 1), instantiated with a EUF-CMA secure signature scheme, the valid() function shown in Algorithm 7, and the* identify() *function described in Algorithm 8, is a responsive secure pod (Definition 11) with Confirmation within $u = 2\delta$, Past-perfection within $w = \delta$ and β-accountable safety (Definition 2), except with negligible probability.*

Proof. Shown in Appendix B. ◀

5 Evaluation

To validate our theoretical results regarding optimal latency in Protocol pod-core, we implement[1] a prototype pod-core in Rust 1.85. Our benchmarks measure the end-to-end confirmation latency of a transaction from the moment it is written by client until it is read as confirmed by another client in a different continent, both interacting with replicas distributed around the world. Specifically, the latency is computed as the difference between the timestamp recorded by the reading client upon receiving sufficiently many votes (quorum size α) from different replicas and the initial timestamp recorded by the writing client. We present the results in Figure 4.

The implementation follows a client-server architecture where each replica maintains two TCP listening sockets: one for the reading client connection and one for the writing client connection. Upon receiving a transaction payload from a writer, the replica creates a tuple containing the payload, a sequence number, and the current local timestamp. The replica then signs this tuple using a Schnorr signature[2] on secp256k1 curve, appends it to its local log, and forwards the signed tuple to the reading client. Replicas are deployed round-robin across seven AWS regions: Frankfurt, London, N. Virginia, N. California, Canada, Mumbai, and Seoul. Each replica is deployed on a t2.medium EC2 instance (2 vCPUs, 4GB RAM) and is initialized with user data that contains the replica's unique secret signing key.

[1] Our prototype implementation is available at `https://github.com/commonprefix/pod-experiments`
[2] `https://crates.io/crates/secp256k1`

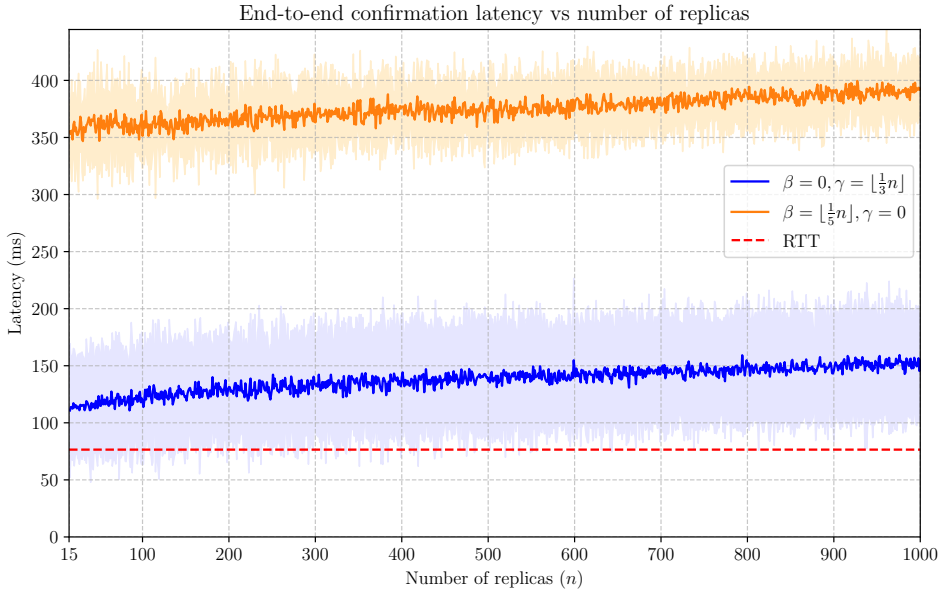


Figure 4 End-to-end confirmation latency from a writing client to a reading client as a transaction traverses across $n = 15, \ldots, 1000$ replicas, for two reading clients: (1) a client that expects up to $\gamma = \lfloor \frac{1}{3} n \rfloor$ omission faults (blue line, below), and (2) a client that expects up to $\beta = \lfloor \frac{1}{5} n \rfloor$ Byzantine faults (orange line, above). The physical network round-trip time between the reading client and the writing client is also shown (dashed red line, 76ms). A 95% confidence interval is shown (shaded).

We implement two types of clients. The writing client establishes connections to all replicas, records the timestamp (in its local view) right before sending the transaction and sends transaction payloads to each replica. The reading client maintains connections to all replicas, validates incoming signed transactions, and records the timestamp (in its local view) upon receiving a quorum of valid signatures for a particular transaction. We deploy the reading client in London and the writing client in N. Virginia, both initialized with the complete list of replica information (IP addresses, public keys).

We conduct experiments with two different values for the quorum size $\alpha = 1 - \beta - \gamma$: (1) $\beta = 0$ and $\gamma = \lfloor \frac{1}{3} n \rfloor$, for a client that only expects omission faults, and (2) $\beta = \lfloor \frac{1}{5} n \rfloor$ and $\gamma = 0$, for a client that expects Byzantine faults. We repeat the experiments for different numbers of replicas ($n = 15, \ldots, 1000$). We repeat each experiment five times and report the mean latency and a 95% confidence interval.

As shown in Figure 4, our experimental results demonstrate that the latency remains largely independent of the number of replicas. The reading client reports a transaction as confirmed as soon as the fastest α replicas have responded, which gives rise to the happy artifact that the 1 - α slowest replicas do not slow down confirmation. This also explains why the omission-fault experiment exhibits lower latency than the Byzantine experiment. Even with 1000 replicas the mean confirmation latency is 138ms for the omission-fault experiment and 375ms for the Byzantine experiment. This approximates the physical network round-trip time between the reading client and the writing client that stands at 76ms.

6 Auctions on pod through the bidset protocol

In this section, we show how single-shot distributed auctions can be implemented on top of pod. This is achieved through bidset, a primitive for collecting a set of bids. The idea is as follows. A pre-appointed *sequencer* – which can be any party, even a pod replica – runs the auction, but the bids are collected from pod using a bidset protocol. The past-perfection property of pod renders the sequencer unable to censor bids: when it creates an output, all timely and honestly-written bids *must* be in it, otherwise the sequencer has provably misbehaved and can be held accountable.

▶ **Definition 16** (bidset protocol). *A bidset* protocol *has a* starting time *parameter* t_0 *and exposes the following interfaces to* bidder *and* consumer *parties:*

- *function submitBid(b): It is called by a bidder at round* t_0 *to submit a bid* b*.*
- *event result*(B, C_{bid})*: It is an event generated by a consumer. It contains a* bid-set B*, which is a set of bids, and auxiliary information* C_{bid}*.*

A bidset protocol satisfies the following liveness and safety properties:

(Liveness) Termination within W: An honest consumer generates an event *result*(B, C_{bid}) by round $t_0 + W$.

(Safety) Censorship resistance: If an honest bidder calls *submitBid*(b) and an honest consumer generates an event *result*$(B, \cdot)$, then $b \in B$.

(Safety) Weak consistency: If two honest consumers generate *result*$(B_1, \cdot)$ and *result*$(B_2, \cdot)$ events, such that $B_1 \neq \emptyset$ and $B_2 \neq \emptyset$, then $B_1 = B_2$.

▶ **Protocol 2** (bidset-core). *Construction bidset-core is parameterized by an integer* Δ *(looking ahead, we will prove security in synchrony, i.e., assuming the network delay* δ *is smaller than* Δ*) and assumes digital signatures and a pod with* δ*-timeliness,* $w = \delta$ *and* $u = 2\delta$*. At time* t_0*, bidders execute Algorithm 4, the sequencer Algorithm 5, and the consumers Algorithm 6.*

Algorithm 4 bidset-core: Code for a bidder. It runs a client for a pod-core instance *pod*.

```
1: function submitBid(b)
2:     pod.write(b)
3: end function
```

Algorithm 5 bidset-core: Code for the sequencer. It runs a client for a pod-core instance *pod*, and sk_a denotes the secret key of the sequencer.

```
1: function readBids()
2:     ((T, r_perf), (C_pp, C_tx)) ← pod.read()
3:     while r_perf ≤ t_0 + Δ do: ((T, r_perf), (C_pp, C_tx)) ← pod.read()
4:     B ← {tx | (tx, ·, ·, ·) ∈ T}; C_bid ← C_pp
5:     σ ← Sign(sk_a, (B, C_bid))
6:     tx ← ⟨BIDS (B, C_bid, σ)⟩
7:     pod.write(tx)
8: end function
```

A bidder (Algorithm 4) submits a bid by writing it on the pod at round t_0. The sequencer (Algorithm 5) waits until the pod returns a past-perfect round larger than $t_0 + \Delta$ (Algorithm 5) and then constructs the bid-set B from the set of transactions in T (Algorithm 5). The sequencer concludes by signing B and C_{bid} (which can be used as evidence, in case of a safety violation) and writing $\langle BIDS\ (B, C_{\text{bid}}, \sigma)\rangle$ on *pod*. The consumer (Algorithm 6) waits until

Algorithm 6 bidset-core: Code for a consumer. It runs a client for a pod-core instance *pod*.

```
1: function readResult()
2:     loop
3:         ((T, r_perf), (C_pp, C_tx)) ← pod.read()
4:         if ∃(tx, ·, ·, r_conf, ·) ∈ T : tx = ⟨BIDS (B, C_bid, σ)⟩ and r_conf ≤ t_0 + 3Δ then
5:             output event result(B, C_bid)
6:         else if r_perf > t_0 + 3Δ then
7:             output event result(∅, C_pp)
8:         end if
9:     end loop
10: end function
```

one of the following two conditions is met. First, a *confirmed* transaction $\langle BIDS\ (B, C_{\text{bid}}, \sigma)\rangle$ appears in T, for which $\mathsf{r}_{\text{conf}} \leq t_0 + 3\Delta$ (Algorithm 6), in which case it outputs bid-set B as result. Second, a round higher than $t_0 + 3\Delta$ becomes past-perfect in pod (Algorithm 6) without a confirmed $\langle BIDS\ \rangle$ transaction appearing, in which case it outputs $B = \emptyset$.

▶ **Theorem 17** (Bidset security)**.** *Assuming a synchronous network where $\delta \leq \Delta$, protocol bidset-core (Construction 2) instantiated with a digital signature and a secure pod protocol that satisfies the* past-perfection within $w = \delta$, confirmation within $u = 2\delta$ *and* δ-timeliness *properties, is a secure bidset protocol satisfying* termination within $W = 3\Delta + \delta$*. It satisfies accountable safety with an* identifySequencer() *function that identifies a malicious sequencer.*

Proof. Additional intuition, formal proofs, and identifySequencer() are shown in the full version of this paper [2, Appendix D]. ◀

▶ Remark 18. Observe that bidset-core terminates within $W = 3\Delta + \delta$ in the worst case, but, if the sequencer is honest, then it terminates within $W = \Delta + 3\delta$. Moreover, bidset-core is not responsive because Algorithm 5 waits for a fixed Δ interval. This step can be optimized if the set of bidders is known (e.g., requiring them to pre-register), which allows for the protocol to be made optimistically responsive (*i.e.*, $W = 4\delta$) when all parties are honest.

▶ Remark 19 (Implicit sub-session identifiers). We assume that each instance of the bidset-core protocol is identified by a unique sub-session identifier (ssid). All messages written to the underlying pod are concatenated with the ssid.

7 Conclusion

In this work we present pod, a novel consensus layer that finalizes transactions with the optimal one-round-trip latency by eliminating communication among replicas. Instead, clients read the system state by performing lightweight computation on logs retrieved from the replicas. As no validator has a particular role in pod (as compared to leaders, block proposers, miners, etc. in similar protocols), pod achieves censorship resistance by default, without any extra mechanisms or additional cost. Furthermore, validator misbehavior, such as voting in incompatible ways or censoring confirmed transactions, is accountable.

As an application, we present an efficient and censorship-resistant mechanism for single-shot first-price and second-price open auctions, which leverages pod as a bulletin board. We show how the accountability, offered by pod, is also inherited by applications built on it – the auctioneer cannot censor confirmed bids without being detected. In the full version [2] of this paper we discuss further applications, including payments in the style of Fastpay [5].

We conjecture that single-shot sealed bid auction protocols, such as those of [12, 4, 9, 24, 6, 13], can also be instantiated on top of a bidset protocol. Intuitively, this holds because such protocols first agree on a set of sealed bids and then determine the winner. A formal analysis of sealed-bid auction protocols based on bidset is left as future work. The architecture of pod also motivates the design of light clients, which would not have to connect to all replicas or to download all transactions. This is achieved through cryptographic primitives such as Merkle mountain ranges and segment trees. We leave the formal description as future work.

Finally, we remark that pod differs from standard notions of consensus because it does not offer an agreement property, neither to validators nor to clients. A client reading the pod obtains a past-perfect round r_{perf}, and it is guaranteed to have received all transactions that obtained a confirmed round r_{conf} such that $\mathsf{r}_{\text{conf}} \leq \mathsf{r}_{\text{perf}}$, or that may obtain such an r_{conf} in the future, even though the transaction presently appears to the client as unconfirmed. However, the client cannot tell which unconfirmed transactions will become confirmed. Moreover, a transaction might appear confirmed to one client and unconfirmed to another (in this case, this is a transaction written by a malicious client).

References

1 Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999. doi:10.1016/S0020-0190(99)00100-3.

2 Orestis Alpos, Bernardo David, and Dionysis Zindros. Pod: An optimal-latency, censorship-free, and accountable generalized consensus layer. *CoRR*, abs/2501.14931, 2025. doi:10.48550/arXiv.2501.14931.

3 Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Low-latency DAG consensus with fast commit path. *CoRR*, abs/2310.14821, 2023. doi:10.48550/arXiv.2310.14821.

4 Samiran Bag, Feng Hao, Siamak F. Shahandashti, and Indranil Ghosh Ray. Seal: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 15:2042–2052, 2020. doi:10.1109/TIFS.2019.2955793.

5 Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *AFT*, pages 163–177. ACM, 2020. doi:10.1145/3419614.3423249.

6 Tarun Chitra, Matheus V. X. Ferreira, and Kshitij Kulkarni. Credible, Optimal Auctions via Public Broadcast. In Rainer Böhme and Lucianna Kiffer, editors, *6th Conference on Advances in Financial Technologies (AFT 2024)*, volume 316 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.AFT.2024.19.

7 Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 26–38. IEEE, 2020. doi:10.1109/DSN48063.2020.00023.

8 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

9 Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. FAST: Fair auctions via secret transactions. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22International Conference on Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 727–747. Springer, Cham, June 2022. doi:10.1007/978-3-031-09234-3_36.

10 Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader BFT via optimistic proposals. *CoRR*, abs/2401.01791, 2024. doi:10.48550/arXiv.2401.01791.

11 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988. doi:10.1145/42282.42283.

12 Hisham S. Galal and Amr M. Youssef. Trustee: Full privacy preserving vickrey auction on top of ethereum. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security*, pages 190–207, Cham, 2020. Springer International Publishing.

13 Chaya Ganesh, Shreyas Gupta, Bhavana Kanukurthi, and Girisha Shankar. Secure vickrey auctions with rational parties. Cryptology ePrint Archive, Paper 2024/1011, 2024. To appear at CCS 2024. doi:10.1145/3658644.3670311.

14 Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *FOCS*, pages 658–668. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.44.

15 Peter Gazi, Ling Ren, and Alexander Russell. Practical settlement bounds for longest-chain consensus. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 107–138. Springer, 2023. doi:10.1007/978-3-031-38557-5_4.

16 Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022. doi:10.1007/978-3-031-18283-9_14.

17 Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. doi:10.1137/0217017.

18 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. *Distributed Comput.*, 35(1):1–15, 2022. doi:10.1007/S00446-021-00399-2.

19 Dahlia Malkhi and Kartik Nayak. Extended abstract: Hotstuff-2: Optimal two-phase responsive BFT. *IACR Cryptol. ePrint Arch.*, page 397, 2023. URL: https://eprint.iacr.org/2023/397.

20 Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 541–559. Springer, Cham, May 2022. doi:10.1007/978-3-031-18283-9_27.

21 Barbara Simons. An overview of clock synchronization. In Barbara Simons and Alfred Spector, editors, *Fault-Tolerant Distributed Computing*, pages 84–96, New York, NY, 1990. Springer New York.

22 Jakub Sliwinski and Roger Wattenhofer. ABC: asynchronous blockchain without consensus. *CoRR*, abs/1909.10926, 2019. arXiv:1909.10926.

23 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version. *CoRR*, abs/2209.05633, 2022. doi:10.48550/arXiv.2209.05633.

24 Nirvan Tyagi, Arasu Arun, Cody Freitag, Riad Wahby, Joseph Bonneau, and David Mazières. Riggs: Decentralized sealed-bid auctions. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1227–1241. ACM Press, November 2023. doi:10.1145/3576915.3623182.

25 Apostolos Tzinas, Srivatsan Sridhar, and Dionysis Zindros. On-chain timestamps are accurate. Cryptology ePrint Archive, Report 2023/1648, 2023. URL: https://eprint.iacr.org/2023/1648.

26 Josef Widder. Booting clock synchronization in partially synchronous systems. In Faith Ellen Fich, editor, *Distributed Computing*, pages 121–135, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/978-3-540-39989-6_9.

27 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

A Validation function for pod-core

In this section we present the function *valid*(), which allows a pod client, which is not necessarily communicating with the pod replicas, to verify that a given pod data structure D satisfies the security properties of pod (Definition 11).

Algorithm 7 Function *valid*(D, C) for pod-core. Code for a *verifier*, which can be a pod client not communicating with the pod replicas.

```
1:  State: Same as in Algorithm 2, includes {R_1, ..., R_n}, tsps, mrt.

2:  function valid(D, C)
3:      (C_pp, C_tx) ← C                          ▷ C_pp : R → vote, C_tx : tx → C_tx, C_tx : R → vote
4:      initState()                                ▷ shown in Algorithm 2
5:      allVotes ← ⋃_{tx∈C_tx} (C_tx[tx].values())
6:      for (tx, ts, sn, σ, R_j) ∈ allVotes in increasing order of sn do
7:          require processVote(tx, ts, sn, σ, R_j)   ▷ shown in Algorithm 2, updates tsps, mrt
8:      end for
9:      require D.T = computeTxSet(tsps, mrt)      ▷ shown in Algorithm 3
10:     require D.r_perf = computePastPerfectRound(mrt)   ▷ shown in Algorithm 3
11:     for (tx, ts, sn, σ, R_j) ∈ C_pp.values() do
12:         require (tx, ts, sn, σ, R_j) ∈ allVotes
13:         require sn = max_sn'((·, ·, sn', ·, R_j) ∈ allVotes)
14:     end for
15: end function
```

The function *valid*() for pod-core is shown in Algorithm 7. The idea is to have the verifier repeat the logic of an honest client. The verifier is initialized in the same way as in Algorithm 2 – importantly, it knows the identifiers and public keys of pod replicas. Function *valid*() takes as input a pod data structure D and auxiliary data C, which contains two parts, a *past-perfection certificate* C_{pp} and a collection of *transaction certificates* $\mathbb{C}_{\text{tx}}$, one for each transaction in $D.\mathsf{T}$. Both contain vote messages, as constructed by a pod client in lines 13 and 14 of Algorithm 2. The verifier processes each vote in order of increasing sequence number sn using function *processVote*(). If any vote is invalid, *valid*() returns false. Observe that if the votes are valid the verifier will have updated its local tsps and mrt variables with the same values as the pod client that constructed D. Finally, the verifier computes the transaction set T and the past-perfect round r_{perf} (using its local tsps and mrt variables) and requires that the values match the ones in D (lines 9–10).

Finally, the verifier also verifies the past-perfection certificate. Given that the previous checks have passed, we require that each vote in C_{pp} is contained in one of the transaction certificates in $\mathbb{C}_{\text{tx}}$ and has the maximum sequence number received from the client that sent the vote (lines 11–14). As we have remarked earlier, C_{pp} can be derived from $\mathbb{C}_{\text{tx}}$ by taking the union of certificates $\mathbb{C}_{\text{tx}}$ for all transactions and keeping the most recent vote for each replica, in which case the checks on lines 11–14 can be omitted. We maintain the past-perfection certificate for readability and simplicity in the proofs.

B Security of Protocol pod-core under a Continuum of Byzantine and Omission faults

In order to prove Theorem 15 and establish the security of Protocol pod-core shown Construction 1, we first prove some useful intermediate results. We remind that $n = \alpha + \beta + \gamma$, where n denotes the total number of replicas, β denotes the number of Byzantine replicas, γ denotes the number of omission-faulty replicas in an execution, and α denotes the number of replicas required to confirm a transaction.

▶ **Lemma 20** (The values for minimum, maximum and confirmed rounds)**.** *Regarding Algorithm 3, we have the following. Consider the list of all timestamps received by a client for a particular transaction, replacing a missing vote from R_j with a special value ($\mathsf{mrt}[R_j]$ for computing r_{min}, ∞ for computing r_{max}), to get n values in total, sorted in increasing order. Assume mrt is also sorted in increasing order of timestamps.*

1. *r_{min} is the timestamp at index $\lfloor \alpha/2 \rfloor - \beta$ of this list.*
2. *r_{max} is the timestamp at index $n - \alpha + \lfloor \alpha/2 \rfloor + \beta$ of this list.*
3. *r_{perf} is the timestamp at index $\lfloor \alpha/2 \rfloor - \beta$ of mrt.*

Proof. Functions *minPossibleTs*() and *computePastPerfectRound*() prepend β times the 0 value in the beginning of the list and return the median of the first α values, hence they return the timestamp at index $\lfloor \alpha/2 \rfloor - \beta$. Function *maxPossibleTs*() appends β times the ∞ value at the end of the list and returns the median of the last α values of that list, that is, it ignores the first $n - \alpha + \beta$ values and returns the timestamp at index $n - \alpha + \beta + \lfloor \alpha/2 \rfloor$. ◀

▶ **Lemma 21** ($\mathsf{r}_{\mathsf{perf}}$ bounded by honest timestamp)**.** *Assuming $n \geq 5\beta + 3\gamma + 1$ (equiv., $\alpha \geq 4\beta + 2\gamma + 1$), for a valid pod D with auxiliary data $C = (C_{pp}, \mathbb{C}_{tx})$, there exists some honest replica R_j, such that the most-recent timestamp mrt from R_j included in C_{pp} satisfies $\mathsf{mrt} \leq D.r_{perf}$.*

Proof. Since *valid*(D, C) = true, the past-perfect round $D.\mathsf{r}_{\mathrm{perf}}$ is the value returned by *computePastPerfectRound*() of Algorithm 3. From Lemma 20 we have that $\mathsf{r}_{\mathrm{perf}}$ is the timestamp at index $\lfloor \alpha/2 \rfloor - \beta$ of sorted mrt. The condition $\alpha \geq 4\beta + 2\gamma + 1$ implies that $\beta + \gamma \leq \lfloor \alpha/2 \rfloor - \beta$, hence the number of not honest replicas $(\beta + \gamma)$ cannot fill all positions between 0 and $\lfloor \alpha/2 \rfloor - \beta$, hence at least one of the indexes between 0 and $\lfloor \alpha/2 \rfloor - \beta$ (inclusive) will contain the timestamp created and sent by an honest replica. ◀

We now recall Theorem 15, which we prove through a series of lemmas.

▶ **Theorem 15** (pod-core security)**.** *Assume that the network is partially synchronous with actual network delay δ, that β is the number of Byzantine replicas, γ the number of omission-faulty replicas, $\alpha = n - \beta - \gamma$ the confirmation threshold, and $n \geq 5\beta + 3\gamma + 1$ the total number of replicas. Protocol pod-core (Protocol 1), instantiated with a EUF-CMA secure signature scheme, the* valid() *function shown in Algorithm 7, and the* identify() *function described in Algorithm 8, is a responsive secure pod (Definition 11) with Confirmation within $u = 2\delta$, Past-perfection within $w = \delta$ and β-accountable safety (Definition 2), except with negligible probability.*

Proof. Follows from Lemmas 22–26, presented and proven in the remainder of this section. ◀

▶ **Lemma 22** (Confirmation within u)**.** *For the conditions stated in Theorem 15, Protocol 1 satisfies the* confirmation within u *property (Definition 11) for $u = 2\delta$.*

Proof. Assume an honest client c calls *write*(tx) at round r. It sends message $\langle \mathit{WRITE}\ \mathsf{tx}\rangle$ to all replicas at round r (line 18). An honest replica receives this by round $\mathsf{r}+\delta$ and sends a $\langle \mathit{VOTE}\ \rangle$ message back to all connected clients (line 20). An honest client c' receives the vote by round $\mathsf{r}+2\delta$. As are at least α honest replicas, c' receives at least α such votes, hence the condition in line 6 is satisfied and c' observes tx as confirmed. ◀

▶ **Lemma 23** (Past-perfection within w)**.** *For the conditions stated in Theorem 15, Protocol 1 satisfies the* past-perfection within *w property (Definition 11) for $w = \delta$.*

Proof. Assume an honest client c at round r has view D^c_{r}. From Lemma 21, there exists some honest replica R_j, such that the most-recent timestamp $\mathsf{mrt}[R_j]$ that R_j has sent to c satisfies $D^c_{\mathsf{r}}.\mathsf{r}_{\text{perf}} \geq \mathsf{mrt}[R_j]$. The honest replica R_j sends at least one heartbeat or vote message per round (line 25), which arrives within δ rounds, and an honest client updates $\mathsf{mrt}[R_j]$ when it receives the heartbeat or vote message. Hence, c will have $\mathsf{mrt}[R_j] \geq \mathsf{r} - \delta$. All together, $D^c_{\mathsf{r}}.\mathsf{r}_{\text{perf}} \geq \mathsf{r} - \delta$. ◀

▶ **Lemma 24** (Past-perfection safety)**.** *For the conditions stated in Theorem 15, Protocol 1 satisfies the* past-perfection *safety property (Definition 11), except with negligible probability.*

Proof. Assume the adversary outputs valid (D_1, C_1) and (D_2, C_2) that violate the property, i.e., there exists a transaction tx such that $(\mathsf{tx}, \mathsf{r}^1_{\text{min}}, \mathsf{r}^1_{\text{max}}, \mathsf{r}^1_{\text{conf}}) \notin D_1.\mathsf{T}$ and $(\mathsf{tx}, \mathsf{r}^2_{\text{min}}, \mathsf{r}^2_{\text{max}}, \mathsf{r}^2_{\text{conf}}) \in D_2.\mathsf{T}$ and $\mathsf{r}^2_{\text{conf}} \neq \perp$ and $\mathsf{r}^2_{\text{conf}} < D_1.\mathsf{r}_{\text{perf}}$. Let $C_1 = (C^1_{\text{pp}}, \mathbb{C}^1_{\text{tx}})$ and $C_2 = (C^2_{\text{pp}}, \mathbb{C}^2_{\text{tx}})$.

Let $\mathcal{R}_1$ be the set of replicas R_i for which C^1_{pp} contains a vote with timestamp $\mathsf{mrt}_i \geq D_1.\mathsf{r}_{\text{perf}}$. From Lemma 20 ($\mathsf{r}_{\text{perf}}$ is computed as the timestamp at index $\lfloor \alpha/2 \rfloor - \beta$ of sorted mrt), and since D_1 is valid, there exist at least $n - \lfloor a/2 \rfloor + \beta$ such replicas, hence $|\mathcal{R}_1| \geq n - \lfloor a/2 \rfloor + \beta$. For each $R_i \in \mathcal{R}_1$, the transaction certificates $\mathbb{C}^1_{\text{tx}}$ contain the whole log of R_i with timestamps up to mrt_i (line 31 of Algorithm 2 does not allow gaps in the sequence number of the received votes). That is, for each $R_i \in \mathcal{R}_1$ the certificates $\mathbb{C}^1_{\text{tx}}$ contains votes

$$(\mathsf{tx}_{i,1}, \mathsf{ts}_{i,1}, 1, \sigma_{i,1}, R_i), (\mathsf{tx}_{i,2}, \mathsf{ts}_{i,2}, 2, \sigma_{i,2}, R_i), \ldots, (\mathsf{tx}_{i,k_i}, \mathsf{ts}_{i,k_i}, k_i, \sigma_{i,k_i}, R_i), \tag{1}$$

where k_i is the smallest sequence number for which $\mathsf{ts}_{i,k_i} \geq D_1.\mathsf{r}_{\text{perf}}$, and $\mathsf{tx}_{i,j}$ are transactions.

Since tx is confirmed in D_2 and $\mathsf{r}^2_{\text{conf}} < D_1.\mathsf{r}_{\text{perf}}$, the transaction certificate $\mathbb{C}^2_{\text{tx}}[\mathsf{tx}]$ must contain votes on tx with timestamp ts_i, such that $\mathsf{ts}_i < D_1.\mathsf{r}_{\text{perf}}$, from at least $\lfloor \alpha/2 \rfloor + 1$ replicas. Let $\mathcal{R}_2$ be the set of these replicas, with $|\mathcal{R}_2| \geq \lfloor \alpha/2 \rfloor + 1$. For each $R_i \in \mathcal{R}_2$, certificate $\mathbb{C}^2_{\text{tx}}[\mathsf{tx}]$ contains a vote

$$(\mathsf{tx}, \mathsf{ts}_i, \mathsf{sn}_i, \sigma_i, R_i), \tag{2}$$

such that $\mathsf{ts}_i < D_1.\mathsf{r}_{\text{perf}}$. We will show that, if at most β replicas are Byzantine, this leads to a contradiction. Observe from the cardinality of $\mathcal{R}_1$ and $\mathcal{R}_2$ that at least $\beta + 1$ replicas must be in both sets, hence at least one honest replica must be in both sets (except if the adversary forges a signature under the public key of an honest replica, which happens with negligible probability). For that replica, the vote in (2) must be one of the votes in (1) since $\mathsf{ts}_i < D_1.\mathsf{r}_{\text{perf}}$ and $\mathsf{ts}_{i,m_i} \geq D_1.\mathsf{r}_{\text{perf}}$. Hence, one of the $\mathsf{tx}_{i,j}$ in (1) is tx, and tx must appear in $D_1.\mathsf{T}$, a contradiction. ◀

▶ **Lemma 25** (Confirmation bounds)**.** *For the conditions stated in Theorem 15, Protocol 1 satisfies the* confirmation bounds *property (Definition 11), except with negligible probability.*

Algorithm 8 The identify() function for Protocol pod-core (Protocol 1).

```
1:  function identify(T)
2:      R̃ ← ∅
3:      for ⟨VOTE (tx_1, ts_1, sn_1, σ_1, R_1)⟩ ∈ T do
4:          if not Verify(pk_1, (tx_1, ts_1, sn_1), σ_1) then continue
5:          for ⟨VOTE (tx_2, ts_2, sn_2, σ_2, R_2)⟩ ∈ T do
6:              if not Verify(pk_2, (tx_2, ts_2, sn_2), σ_2) then continue
7:              if R_1 = R_2 and sn_1 = sn_2 and (tx_1 ≠ tx_2 or ts_1 ≠ ts_2) then
8:                  R̃ ← R̃ ∪ {R_1}
9:              end if
10:         end for
11:     end for
12: end function
```

Proof. Assume the adversary outputs (D_1, C_1) and (D_2, C_2), such that $\mathit{valid}(D_1, C_1) \wedge \mathit{valid}(D_2, C_2)$ and there exists a transaction tx such that $(\mathsf{tx}, \mathsf{r}^1_{\min}, \mathsf{r}^1_{\max}, \mathsf{r}^1_{\text{conf}}) \in D_1.\mathsf{T}$ and $(\mathsf{tx}, \mathsf{r}^2_{\min}, \mathsf{r}^2_{\max}, \mathsf{r}^2_{\text{conf}}) \in D_2.\mathsf{T}$. Let $C_1 = (C^1_{\text{pp}}, \mathbb{C}^1_{\text{tx}})$ and $C_2 = (C^2_{\text{pp}}, \mathbb{C}^2_{\text{tx}})$, and $C^1_{\text{tx}} = \mathbb{C}^1_{\text{tx}}[\mathsf{tx}]$ and $C^2_{\text{tx}} = \mathbb{C}^2_{\text{tx}}[\mathsf{tx}]$.

First assume $\mathsf{r}^1_{\min} > \mathsf{r}^2_{\text{conf}}$. From Lemma 20, C^1_{tx} can include at most $\lfloor \alpha/2 \rfloor - \beta$ votes with a timestamp for tx smaller than $\mathsf{r}^1_{\min}$. Allowing up to β replicas to equivocate, the adversary can obtain at most $\lfloor \alpha/2 \rfloor$ votes on tx with a timestamp smaller than $\mathsf{r}^1_{\min}$, except if it forges a digital signature from an honest replica, which happens with negligible probability. In order to compute $\mathsf{r}^2_{\text{conf}} < \mathsf{r}^1_{\min}$ for tx, the adversary must include in C^2_{tx} timestamps smaller than $\mathsf{r}^1_{\min}$ from at least $\lfloor \alpha/2 \rfloor + 1$ replicas.

Now assume $\mathsf{r}^1_{\max} < \mathsf{r}^2_{\text{conf}}$. Using Lemma 20, C^1_{tx} can include at most $\alpha - \lfloor \alpha/2 \rfloor - \beta - 1$ votes with a timestamp larger than $\mathsf{r}_{\max}$, hence the number of honest replicas, from which a vote with timestamp larger than $\mathsf{r}_{\max}$ can be included in C^2_{tx} is at most $\alpha - \lfloor \alpha/2 \rfloor - 1$ (since β are malicious). If α is odd, this upper bound becomes $\alpha - \lfloor \alpha/2 \rfloor - 1 = \lfloor \alpha/2 \rfloor$, while at least $\lfloor \alpha/2 \rfloor + 1$ votes larger that $\mathsf{r}_{\max}$ are required to compute a median larger than $\mathsf{r}_{\max}$, and if α is even, then $\alpha - \lfloor \alpha/2 \rfloor - 1 = \lfloor \alpha/2 \rfloor - 1$, while at least $\lfloor \alpha/2 \rfloor$ votes larger that $\mathsf{r}_{\max}$ are required to compute a median larger than $\mathsf{r}_{\max}$. (We remind that Algorithm 3 returns as median the value at position $\lfloor \alpha/2 \rfloor$). In either case, we get a contradiction, except for the negligible probability that the adversary forges a digital signature from an honest replica. ◀

▶ **Lemma 26** (β-Accountable safety). *For the conditions stated in Theorem 15, Protocol 1 satisfies accountable safety (Definition 2) with resilience β, except with negligible probability.*

Proof. We show that identify() (Algorithm 8) satisfies the *correctness* and *no-framing* properties required by Definition 2, in three steps.

1. If the past-perfection safety property (Definition 11) is violated, there exists a partial transcript T, such that identify() on input T returns at least β replicas.
Proof: We resume the proof of Lemma 24. There, we constructed sets $\mathcal{R}_1, \mathcal{R}_2$, such that $\mathcal{R}_1 \cap \mathcal{R}_2 \geq \beta + 1$. We saw that, for each $R_i \in \mathcal{R}_1 \cap \mathcal{R}_2$, certificates $\mathbb{C}^1_{\text{tx}}$ contain the replica log shown in (1), containing all votes with timestamp up to $\mathsf{ts}_{i,k_i} \geq \mathsf{r}_{\text{perf}}$. In a similar logic, certificates $\mathbb{C}^2_{\text{tx}}$ contains the following k'_i votes from R_i (possibly more, but we care for the votes up to transaction tx)

$$(\mathsf{tx}'_{i,1}, \mathsf{ts}'_{i,1}, 1, \sigma'_{i,1}, R_i), (\mathsf{tx}'_{i,2}, \mathsf{ts}'_{i,2}, 2, \sigma'_{i,2}, R_i), \ldots, (\mathsf{tx}'_{i,k'_i}, \mathsf{ts}'_{i,k'_i}, k'_i, \sigma'_{i,k'_i}, R_i), \tag{3}$$

with $\mathsf{tx}'_{i,k'_i} = \mathsf{tx}$ and $\mathsf{ts}'_{i,k'_i} < \mathsf{r}_{\text{perf}}$. Obviously, for an honest R_i, the replica logs of (1) and (3) must be identical, i.e., $\mathsf{tx}_{i,j} = \mathsf{tx}'_{i,j}$ and $\mathsf{ts}_{i,j} = \mathsf{ts}'_{i,j}$, for $j \in [1, \min(k_i, k'_i)]$. We will show that they differ in at least one sequence number. If $k_i > k'_i$, then the replica logs differ at sequence number k'_i, because the transaction tx_{i,k_i} in (1) cannot be tx, as $D_1.\mathsf{T}$ does not contain tx, and $\mathsf{tx}'_{i,k'_i} = \mathsf{tx}$. If $k_i \leq k'_i$, the log of (1) should be identical with the first k_i positions of the log of (3), which would imply that $\mathsf{ts}_{i,k_i} = \mathsf{ts}'_{i,k_i}$ and, since a valid pod only accepts non-decreasing timestamps, $\mathsf{ts}'_{i,k_i} \leq \mathsf{ts}'_{i,k'_i}$, and all together $\mathsf{ts}_{i,k_i} \leq \mathsf{ts}'_{i,k'_i}$. This is impossible, because $\mathsf{ts}_{i,k_i} > \mathsf{r}_{\text{perf}}$ and $\mathsf{ts}'_{i,k'_i} < \mathsf{r}_{\text{perf}}$. Hence, the two logs will contain a different timestamp for some sequence number in $[1, k'_i]$.

Summarizing, we have shown for at least $\beta + 1$ replicas $R_i \in \mathcal{R}_1 \cap \mathcal{R}_2$, certificate C_1 and C_2 contain votes $(\mathsf{tx}_1, \mathsf{ts}_1, \mathsf{sn}_1, \sigma_1, R_i)$ and $(\mathsf{tx}_2, \mathsf{ts}_2, \mathsf{sn}_2, \sigma_2, R_i)$, such that $\mathsf{sn}_1 = \mathsf{sn}_2$ but $\mathsf{tx}_1 \neq \mathsf{tx}_2$ or $\mathsf{ts}_1 \neq \mathsf{ts}_2$. On input a set T that contains these votes, function $\mathsf{identify}(T)$ returns $\mathcal{R}_1 \cap \mathcal{R}_2$.

2. If the confirmation-bounds property (Definition 11) is violated, there exists a partial transcript T, such that Algorithm 8 on input T returns at least β replicas.

Proof: As in the proof of Lemma 25, assume the adversary outputs (D_1, C_1) and (D_2, C_2), such that $\mathit{valid}(D_1, C_1) \wedge \mathit{valid}(D_2, C_2)$ and there exists a transaction tx such that $(\mathsf{tx}, \mathsf{r}^1_{\text{min}}, \mathsf{r}^1_{\text{max}}, \mathsf{r}^1_{\text{conf}}) \in D_1.\mathsf{T}$, $(\mathsf{tx}, \mathsf{r}^2_{\text{min}}, \mathsf{r}^2_{\text{max}}, \mathsf{r}^2_{\text{conf}}) \in D_2.\mathsf{T}$, and $\mathsf{r}^1_{\text{min}} > \mathsf{r}^2_{\text{conf}} \vee \mathsf{r}^1_{\text{max}} < \mathsf{r}^2_{\text{conf}}$ Let $C_1 = (C^1_{\text{pp}}, \mathbb{C}^1_{\text{tx}})$ and $C_2 = (C^2_{\text{pp}}, \mathbb{C}^2_{\text{tx}})$, and $C^1_{\text{tx}} = \mathbb{C}^1_{\text{tx}}[\mathsf{tx}]$ and $C^2_{\text{tx}} = \mathbb{C}^2_{\text{tx}}[\mathsf{tx}]$.

Let's take the case $\mathsf{r}^1_{\text{min}} > \mathsf{r}^2_{\text{conf}}$ first. From Lemma 20 ($\mathsf{timestamps}$ contains at least $n - \lfloor \alpha/2 \rfloor + \beta$ timestamps ts such that $\mathsf{ts} \geq \mathsf{r}_{\text{min}}$), there is a set $\mathcal{R}_1$ with at least $n - \lfloor \alpha/2 \rfloor + \beta$ replicas R_i, from each of which $\mathbb{C}^1_{\text{tx}}$ contains votes

$$(\mathsf{tx}_{i,1}, \mathsf{ts}_{i,1}, 1, \sigma_{i,1}, R_i), (\mathsf{tx}_{i,2}, \mathsf{ts}_{i,2}, 2, \sigma_{i,2}, R_i), \ldots, (\mathsf{tx}_{i,m_i}, \mathsf{ts}_{i,m_i}, m_i, \sigma_{i,m_i}, R_i), \tag{4}$$

up to some sequence number m_i, such that $\mathsf{ts}_{i,m_i} \geq \mathsf{r}_{\text{min}}$ and either $\mathsf{tx}_{i,m_i} = \mathsf{tx}$ (i.e., a vote from R_i on tx is included in C^1_{tx}, and we only consider the votes up to this one), or $\mathsf{tx}_{i,j} \neq \mathsf{tx}, \forall j \leq m_i$ (i.e., a vote from R_i on tx is not included in C^1_{tx}, in which case $\mathsf{timestamps}$ contains the timestamp R_i has sent on $\mathsf{tx}_{i,m_i} \neq \mathsf{tx}$).

Now, for a valid D_2 to output $\mathsf{r}^2_{\text{conf}} < \mathsf{r}^1_{\text{min}}$, certificate C^2_{tx} must contain timestamps smaller than r_{min} from at least $\lfloor \alpha/2 \rfloor + 1$ replicas. Call this set $\mathcal{R}_2$. From each of these replicas, certificates $\mathbb{C}^2_{\text{tx}}$ must contain votes

$$(\mathsf{tx}'_{i,1}, \mathsf{ts}'_{i,1}, 1, \sigma'_{i,1}, R_i), (\mathsf{tx}'_{i,2}, \mathsf{ts}'_{i,2}, 2, \sigma'_{i,2}, R_i), \ldots, (\mathsf{tx}, \mathsf{ts}'_{i,m'_i}, m'_i, \sigma'_{i,m'_i}, R_i), \tag{5}$$

considering only votes up to tx, for which $\mathsf{ts}'_{i,m'_i} < \mathsf{r}_{\text{min}}$.

By counting arguments there are at least $\beta + 1$ replicas in $\mathcal{R}_1 \cap \mathcal{R}_2$. For each one, we make the following argument. Since $\mathsf{ts}_{i,m_i} \geq \mathsf{r}_{\text{min}}$ and $\mathsf{ts}'_{i,m'_i} < \mathsf{r}_{\text{min}}$, we get $\mathsf{ts}'_{i,m'_i} < \mathsf{ts}_{i,m_i}$, and it must be the case that $m'_i < m_i$ (otherwise, the two logs will differ at a smaller sequence number, similar to the previous case). But in this case the two logs differ at sequence number m'_i, i.e., $\mathsf{tx}_{i,m'_i} \neq \mathsf{tx}'_{i,m'_i} = \mathsf{tx}$. This is because the log of (4) either does not contain tx, or contains it at sequence number $m_i > m'_i$, in which case it must contain a different transaction at sequence number m'_i. On input a set T that contains all votes for replicas in $\mathcal{R}_1$ and $\mathcal{R}_2$ votes, function $\mathsf{identify}(T)$ returns $\mathcal{R}_1 \cap \mathcal{R}_2$.

For the case $\mathsf{r}^1_{\text{max}} < \mathsf{r}^2_{\text{conf}}$, similar arguments apply. In order to compute $\mathsf{r}^2_{\text{conf}} > \mathsf{r}^1_{\text{max}}$, certificate C^2_{tx} must contain at least $\lfloor \alpha/2 \rfloor$ or $\lfloor \alpha/2 \rfloor + 1$ (depending on the parity of α) votes on tx with timestamp larger than r_{max}. On the other hand, from Lemma 20 certificate C^1_{tx} contains at least $n - \alpha + \lfloor \alpha/2 \rfloor + \beta$ votes on tx with a timestamp smaller or equal than r_{max}. As before, the replicas in the intersection of these two sets have sent conflicting votes for some sequence numbers.

3. The identify() function never outputs honest replicas.
Proof: The function only adds a replica to $\tilde{R}$ if given as input two vote messages from that replica, where the same sequence number is assigned to two different votes (line 7 on Algorithm 8). An honest replica always increments nextsn after each vote it inserts to its log (line 21 on Algorithm 1), hence, the adversary can only construct valid votes by forging a signature under the public key of an honest replica, which happens with negl. probability. ◀

DAG It Off: Latency Prefers No Common Coins

Ignacio Amores-Sesar
Aarhus University, Denmark

Viktor Grøndal
Aarhus University, Denmark

Adam Holmgård
Aarhus University, Denmark

Mads Ottendal
Aarhus University, Denmark

Abstract

We introduce Black Marlin, the first Directed Acyclic Graph (DAG)-based Byzantine atomic broadcast protocol in a partially synchronous setting that successfully forgoes the reliable broadcast and common coin primitives. Black Marlin achieves the optimal latency of 3 rounds of communication (4.25 with Byzantine faults) while maintaining optimal communication and amortized communication complexities. We present a formal security analysis of the protocol, accompanied by empirical evidence that Black Marlin outperforms state-of-the-art DAG-based protocols in both throughput and latency.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Atomic broadcast, DAG-based, Partial synchrony

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.5

Related Version *arXiv version*: `https://arxiv.org/abs/2508.14716`

Funding This work has been mostly funded by the Cryptographic Foundations for Digital Society, CryptoDigi, DFF Research Project 2, Grant ID 10.46540/3103-00077B, and partially funded by the Swiss National Science Foundation(SNSF) under grant agreement Nr. 188443 (Advanced Consensus Protocols) and by a grant from Avalanche, Inc. to the University of Bern.

1 Introduction

Scalability poses a significant challenge for consensus protocols. Paradoxically, increasing the number of participants in consensus degrades its performance rather than enhancing it.

Traditional consensus protocols, such as Paxos [19], PBFT [10], and HotStuff [27], primarily adopt a leader-based approach. In this model, a designated leader proposes a value, which the remaining participants validate. This design results in an uneven distribution of workload, with the leader becoming a performance bottleneck. If we add more parties to a consensus protocol, we simply increase the communication load of the leader, degrading the performance of the protocol. Furthermore, if the leader behaves in a Byzantine manner, the system must invoke a view-change mechanism to replace it. These view-changes are sensitive to timing assumptions and become progressively more expensive [8] as the number of parties increases.

To address these limitations, Keidar et al. [16] introduced DAG-Rider, an elegant asynchronous solution that leverages the common core abstraction [9]. This approach enables every participant to act as a leader in the protocol. These instances are interwoven into a directed acyclic graph (DAG), on top of which a total ordering is achieved using a common coin [8], applied at the end of every *wave*: a set of four consecutive rounds.


© Ignacio Amores-Sesar, Viktor Grøndal, Adam Holmgård, and Mads Ottendal;
licensed under Creative Commons License CC-BY 4.0



DAG-Rider demonstrates remarkable throughput compared to leader-based protocols, but this comes at the cost of significantly increased latency. The primary cause of this latency is the repeated use of reliable broadcast [8] across each of the four rounds required to implement the common core primitive [9]. This triggered the effort of the research community [12, 26, 17, 15, 24, 6, 25, 5] aimed at mitigating this latency overhead. These contributions generally fall into two broad categories.

On the one hand, Cordial Miners [17] replaces the reliable broadcast rounds with bare-message communications, reducing communication overhead at the cost of increasing the wave length to five rounds (three in partial synchrony). On the other hand, a majority of the works [12, 26, 15, 24, 6, 25, 5] pursue orthogonal improvements, either through more efficient use of the common coin or by shortening the length of the waves under partial synchronous assumptions. These approaches have collectively succeeded in halving the latency of DAG-based protocols, marking a significant advance in the state of the art. However, their reliance on the reliable broadcast primitive continues to be a limiting factor. Moreover, to the best of our knowledge, no existing protocol has yet successfully elected an anchor in every round.

These are precisely the two limitations addressed in this paper. We introduce Black Marlin, the first partially synchronous, DAG-based protocol that elects an anchor in every round without relying on reliable broadcast. The simplest – yet most impactful – innovation is the substitution of the common coin with a deterministic round-robin mechanism. This substitution is valid because the FLP impossibility [14] only applies in asynchronous settings.

The primary advantage of the round-robin approach is that it allows block creation to be conditioned on the reception of the anchor block from a given round. To handle scenarios in which a malicious anchor fails to behave correctly, Black Marlin employs timeouts, thereby preventing deadlocks. This conditional block creation enables Black Marlin to elect an anchor in every round, fulfilling the long-standing goal of the research on DAG-based protocols.

Moreover, by operating in a partially synchronous model, Black Marlin is able to adapt ideas from Cordial Miners [17] to eliminate the need for reliable broadcast, using only digital signatures without the need for extra rounds of communication. The trade-off for achieving anchor election without reliable broadcast is a small constant delay and a minor reduction in commitment probability under Byzantine behavior, while maintaining optimal message complexity in both cases.

Specifically, Black Marlin commits the anchor of round $r - 2$ upon completing round r with probability $4/9$ under adversarial conditions. This probability increases to 1 when all participants behave honestly. As a result, the expected latency is 4.25 communication rounds under Byzantine behavior and the optimal 3 rounds in the honest case.

Additionally, timeouts are only triggered during asynchronous periods or when facing Byzantine faults. Thus, Black Marlin remains a reactive protocol, effectively leveraging the advantages of both synchronous and asynchronous models. We prove that Black Marlin implements atomic broadcast [8] together with all the performance results mentioned above. Furthermore, we benchmark Black Marlin against Bullshark, showing its practicality.

1.1 Structure

The paper is structured as follows. Section 1 briefly describes the paper. Section 2 details the state of the art of DAG-based protocols. Section 3 sets the notation and key concepts used in the paper. Section 4 describes Black Marlin in full detail. Section 5 shows that Black Marlin implements atomic broadcast and studies its complexities. Section 6 describes a comparison of the throughput and latency of Black Marlin and Bullshark.

2 Related work

DAG-based protocols have constituted a significant transformation in the field of consensus over the past few years.

This revolution began in the permissionless setting, with protocols such as Avalanche [22] and IOTA [20]. However, these protocols were initially introduced without thorough security analyses. As a result, several vulnerabilities were later discovered [11, 4, 3], which limited their success despite recent work demonstrating their optimality [2].

DAG-based consensus protocols were later extended to the permissioned setting, most notably with DAG-Rider by Keidar et al. [16]. DAG-Rider was the first protocol to leverage the properties of the common core primitive [9] to implement atomic broadcast in asynchronous networks. It outperforms traditional leader-based protocols [19, 10, 27] in terms of throughput. However, DAG-Rider requires on average six instances of reliable broadcast [7], resulting in 18 rounds of communication. Since its introduction, several works have aimed to address the latency limitations of DAG-Rider.

On one hand, Cordial Miners [17] reduces latency to an average of 7.5 communication rounds by replacing reliable broadcast with bare-message communication. The tradeoff of this approach is that its new communication paradigm requires an additional round per wave to compensate for the properties that reliable broadcast provides. When a party creates a block in Cordial Miners, it must include any blocks the recipient may have missed. We adopt this idea in the design of Black Marlin.

On the other hand, a widely adopted strategy to improve DAG-Rider's latency has involved either assuming synchronous networks or modifying the core and coin primitives. Narwhal and Tusk [12] introduced a clever transaction preprocessing mechanism, where transactions are grouped into batches, so only the hashes of these batches are submitted to consensus. This batching is delegated to specialized *workers*, allowing the system to scale dynamically with transaction load. However, Narwhal and Tusk lack a rigorous security analysis, and liveness vulnerabilities have been identified [23].

Bullshark [26], which builds on top of Narwhal and Tusk, also introduces a variant for partial synchrony that improves latency to just three instances of reliable broadcast, about nine rounds of communication, on average. Bullshark is particularly relevant to our work, as its authors provide a public implementation [21][1], enabling a fair, side-by-side comparison with Black Marlin. It is worth noting that both Narwhal and Tusk, as well as Bullshark, rely on reliable broadcast as a fundamental building block.

More recently, several protocols have been proposed targeting latency. These works largely follow the above trajectory. Mysticeti [6] reduces latency further by employing consistent broadcast, achieving an average-case latency of only three instances [8]. Sailfish [24] also improves latency, though it continues to rely on reliable broadcast. Similarly, Shoal and Shoal++ [25] provide enhancements in anchor selection, but still depend on reliable broadcast.

To the best of our knowledge, Black Marlin is the first DAG-based protocol to achieve optimal latency in partial synchrony, while electing an anchor in every round, a crucial feature for minimizing the latency of non-anchor blocks without relying on reliable broadcast. We also highlight that Black Marlin achieves these properties while preserving simplicity.

[1] Bullshark has been chosen over more recent frameworks such as Mysticeti [18] due to its closer similarity to Black Marlin and the timing of this project.

3 Model

3.1 Notation

We consider a set $\mathcal{N} = \{0, ..., n-1\}$ of n parties running a round-based algorithm. In each round, these parties create blocks that form the vertices of a directed acyclic graph $(DAG, \mathcal{R})$ with references that constitute the edges. Edge $(B_1, B_2) \in \mathcal{R}$ if B_1 includes a reference to B_2. A block $B = [r, i, \mathit{refs}, \mathit{refs}']$ is a tuple formed by a round number r, an identifier for party i, a set of strong references *refs*, and a set of weak references refs'. For simplicity, we omit the set of transactions and the signature from the block, as these transactions play no role in the consensus mechanism. Given a block B, $\mathit{creator}(B)$ denotes the party that created B, $\mathit{round}(B)$ denotes the round in which B was created. The variable $DAG(r)$ denotes the set of block $B' \in DAG$ such that $\mathit{round}(B') = r$. A party selected by a round-robin mechanism is referred to as *anchor*, blocks produced by the anchor party are referred to as *anchor* blocks.

3.2 Adversary and Network

We model parties as *interactive Turing machines* (*ITM*). An interactive Turing machine is a Turing machine with an input and an output tape that allows the Turing machine to communicate with other Turing machines and make decisions based on the content of their input tape. The adversary is modeled as another ITM that corrupts up to f parties at the beginning of the execution, the actual number of corruptions is denoted by $t \leq f < \frac{n}{3}$. These corrupted parties obey the adversary; i.e., they may diverge from the execution of the protocol. Corrupted parties are often referred to as *Byzantine* and non-corrupted as *honest*.

We assume a partial synchronous communication model [13] on point-to-point links. The network begins in an asynchronous state and turns synchronous after a global stabilization time (GST). After this point, every message sent is delivered with a delay of up to Δ. The adversary not only decides the global stabilization time but also when the message is made available to the party within the Δ time limit. We also assume that the local computations are instantaneous, this assumption can easily be relaxed by factoring the computation of the honest parties in the network delay. We assume the existence of a public key infrastructure. We also assume that every block and a message produced by every party is signed, and that the signature scheme is secure.

3.3 Atomic broadcast

We model our protocol as *atomic broadcast*. Our atomic broadcast primitive is accessed with the events $\mathit{ab\text{-}broadcast}(B, i, r)$ and outputs events $\mathit{ab\text{-}deliver}(B, j, r)$. The event $\mathit{ab\text{-}broadcast}(B, i, r)$ can only be triggered by party i, but $\mathit{ab\text{-}deliver}(B, j, r)$ may be output by every party.

▶ **Definition 1** (Atomic broadcast)**.** *A protocol solves* atomic broadcast *if it satisfies the following properties, except with negligible probability:*

Validity. *If an honest party i invokes* ab-broadcast(B, i, r) *after the global stabilization time, then party i eventually outputs* ab-deliver(B, i, r).

Agreement. *If an honest party outputs* ab-deliver(B, i, r)*, then all honest parties eventually output* ab-deliver(B, i, r).

Integrity. *For any party i and round r, every honest party outputs* ab-deliver(B, i, r) *at most once regardless of B. Moreover, if i is honest, then i has invoked* ab-broadcast(B, i, r).

Total order. *If an honest party outputs* ab-deliver(B, i, r) *before* ab-deliver(B', j, r')*, then no honest party outputs* ab-deliver(B', j, r') *before* ab-deliver(B, i, r).

Note that our definition of atomic broadcast is adapted to the partial synchrony model; the validity property applies to blocks that are broadcast after the network becomes synchronous. However, the safety properties are satisfied during both synchrony and asynchrony.

4 Protocol

Black Marlin (Algorithm 2) proceeds in rounds. In round r, honest parties wait to receive blocks from at least $n!-!f$ parties. They then check if an anchor for round r has been received and the anchors for rounds $r-2$ and $r-1$ appear in the past of at least $n-f$ blocks from rounds $r-1$ and r, respectively. If both hold, a new block is created referencing all received blocks, and the party advances to round $r+1$. Otherwise, the party waits until the conditions are met or a timeout triggers block creation. These timeouts ensure liveness under Byzantine anchors and are only used when progress is blocked, allowing the protocol to be responsive. Figure 1 illustrates an example execution.

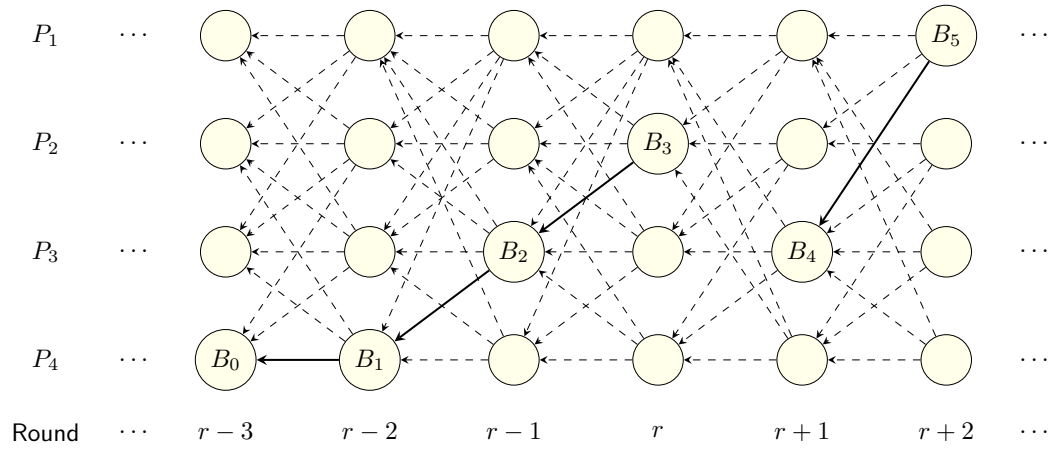


Figure 1 An example of execution of Black Marlin (Algorithm 2) with $f = 1$ and $n = 4$. Every block has references to at least 3 blocks from the previous round. The blocks $B_0, ..., B_5$ are the anchor blocks for the corresponding round. Note that blocks B_0, B_1, B_2, and B_3 have enough support (i.e., at least 3 references) to be committed. However, $B_3 \notin \mathit{strong}(B_4)$, thus B_0, B_1, and B_2, satisfy the commitment rule (L18–32) while B_3 does not. After the global stabilization time, this situation cannot occur without malicious behavior, as B_3 would satisfy $B_3 \in \mathit{past}(B_4)$. This situation has been constructed to illustrate the commitment rule.

4.1 Validity predicates

A block from round $r \neq 1$ and creator j is valid if it is signed by party j and contains references to valid blocks created by at least $n-f$ parties from round $r-1$. Since blocks in round $r = 1$ do not reference any previous blocks, a block is valid if it has been signed by its creator. A history $\mathcal{H}$ is valid if all the blocks contained in it are also valid.

When receiving a message $[\text{BLOCK}, B', \mathcal{H}', j, r']$ (L57), party i checks the validity of both the block B' and the history $\mathcal{H}'$, and adds the block B' along with every unknown block from the history $\mathcal{H}'$ to its local view (DAG). Party i also updates party j's local history (L60), used solely to optimize message complexity by sending only strictly necessary blocks.

4.2 Initialization

The first round of the protocol differs from the rest as there are no blocks from earlier rounds. Party i creates a block with an empty set of references and invokes the event *ab-broadcast*(B, i) before sending the message $[\text{BLOCK}, B, \{\}, i, 0]$ to every party and starting a timeout with time 2Δ. Party i then moves to the next round, a standard round.

4.3 Round structure

Every round beyond the first shares the same structure. Consider an honest party i in round r, i.e., it has just produced a block corresponding to round r. Party i waits to receive messages $[\text{BLOCK}, B, \mathcal{H}, j, r']$ from at least $n - f$ different parties for some round $r' \geq r$. Once sufficient messages for such a round have been received, party i checks whether it has received a message $[\text{BLOCK}, B', \mathcal{H}', j, r']$ with j being the anchor for the current round, and verifies that the previous two anchors have enough support, or if a timeout has been triggered, to conclude the round. This dual condition to start a new round allows the protocol to be responsive, i.e., it runs at the speed of the actual network delay instead of the estimated network delay Δ without deadlocking when an anchor party behaves maliciously.

Party i concludes the round by performing the following sequential operations. First, i attempts the delivery of blocks (L42); this function is described in its own paragraph below. Secondly, party i creates a new block B corresponding to the next round $r' + 1$, including strong references *refs* to every block in $DAG(r')$ (L43–50). A set of weak references *refs*$'$ to blocks from previous rounds are also included; this secondary set is time-bounded, preventing the usual garbage collection problem associated with these solutions. Party i concludes these operations by sending B to every other party, together with the history of the parties (L54). Note that party i may skip one or multiple rounds if an appropriate quorum is received.

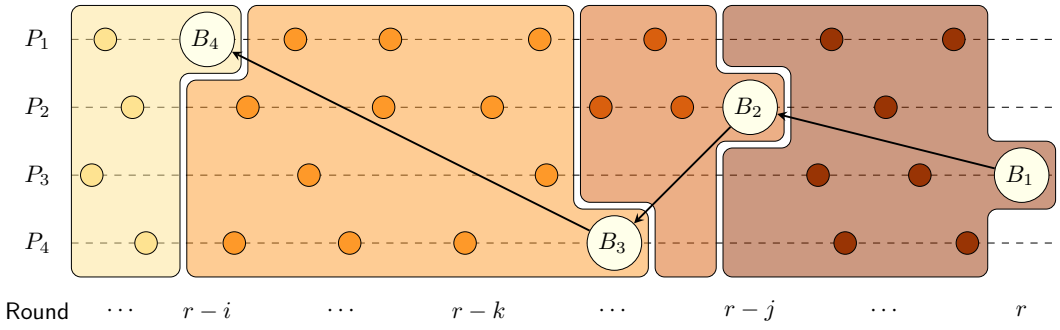


Figure 2 Illustration of the commit rule of Black Marlin. When a party invokes the commit function on an anchor block B_1, it will recursively commit earlier uncommitted anchor blocks first. The subscript numbers on the blocks indicate the order in which the commit function is called. In this process, blocks B_1 B_2, B_3, and B_4 are *ab-delivered*, in the reverse order of their commitment. Before committing any anchor block, all non-delivered blocks in their causal history are deterministically sorted and committed. In the illustration, these correspond to the smaller vertices contained within the history boxes associated with each anchor block.

4.4 Delivery

The delivery function determines which blocks are delivered and in what order, based on the local view of the DAG. Upon concluding round r, party i attempts the commitment of the anchor block B from round $r - 2$ (L14–17). B is an anchor block if its creator has been

elected by the round-robin mechanism. However, it is only committed if it is supported by at least $n - f$ parties in round $r - 1$ and including an anchor of round $r - 1$ B', and B' is also supported by at least $n - f$ parties in round r. These conditions prevent honest parties from committing different blocks when the anchor party is Byzantine.

If this check succeeds, party i calls the function $commit(B)$ (L18–32), which iterates through the local view of the DAG. Party i searches for a non-delivered block B' from a higher round reachable through strong references from B. If such a block B' exists, the function $commit(B')$ is called recursively. If no such block exists, party i deterministically orders the blocks in $past(B)$ before invoking *ab-deliver* on them. In case of conflicting blocks from the same party and round, only the first block – determined by the deterministic ordering above – is *ab-delivered*. Figure 2 shows an example of the commitment rule.

4.5 Functions

Black Marlin (Algorithm 2) uses special functions as described in the paragraphs above. For the sake of the reader, we summarize the functions together in this section.

Algorithm 1 State and functions (party i).

```
1: 𝒩                                                              set of parties
2: r ← 0                                                          current round
3: DAG ← ∅                                                        local view of the DAG
4: 𝒟 ← ∅                                                          set of delivered blocks
5: refs ← ∅                                                       set of strong references
6: refs' ← ∅                                                      set of weak references
7: history[j ∈ 𝒩] ← ∅                                             set of blocks known by party j

8: function quorum(r)
9:     return |parties(DAG(r))| ≥ n − f

10: function anchor(r)
11:     return ∃B ∈ DAG(r) : creator(B) = RR(r)

12: function suppAnchor(r)
13:     return ∃B ∈ DAG(r) : creator(B) = RR(r) ∧ supp(B) ≥ n − f

14: function delivery(r)                                          //attempt a block in r − 2
15:     B ← RR(r − 2)                                             //if multiple consider them all
16:     if supp(B) ≥ n − f ∧ ∃B' ∈ DAG(r − 1) :
            creator(B') = RR(r − 1) ∧ B ∈ strong(B') ∧ supp(B') ≥ n − f then
17:         commit(B)

18: function commit(B)
19:     𝒜 ← strong(B) \ 𝒟                                         //ignore already delivered blocks
20:     if 𝒜 ≠ ∅ then
21:         if |maxAnchor(𝒜)| = 1 then
22:             B' ← maxAnchor(𝒜)                                 //anchor from the highest round
23:         else
24:             B' ← A ∈ maxAnchor(𝒜) : |round(A) − round(maxAnchor(strong(A)))|
                is minimal and break ties deterministically
25:         commit(B')
26:     for B' ∈ τ(past(B) \ 𝒟) do                                // with τ any deterministic topological sorting
27:         if ∄B* ∈ 𝒟 : creator(B') = creator(B*) ∧ round(B') = round(B*) then
28:             ab-deliver(B', creator(B'), round(B'))
29:             𝒟 ← 𝒟 ∪ B'
30:     ab-deliver(B, creator(B), round(B))
31:     𝒟 ← 𝒟 ∪ B
32:     return
```

Given a block B, the function $creator(B)$ returns the party that created and signed B, and the function $round(B)$ returns the round in which B was created. The validity predicate $V(B)$ returns 1 if B has at least $n - f$ blocks from different parties and no two blocks from the same party corresponding to $round(B) - 1$ in its references, and B has been signed by its creator. When invoked with a set of blocks $\mathcal{H}$, the validity predicate $V(\mathcal{H})$ returns 1 if every block $B' \in \mathcal{H}$ satisfies $V(B')$. The function $past(B)$ returns the set of blocks reachable from B through strong and weak links, and the function $strong(B)$ returns the set of blocks reachable from B through strong links, B is not part of the output of either function. The function $supp(B)$ counts the number of parties i that created a block B_i in round $round(B) + 1$ such that $B \in strong(B_i)$ and no other block from the same creator and round is in $strong(B_i)$. The function $H(B)$ is a collision-resistant hash function. The operation $setTimeout(r, \rho)$ starts a timeout of duration ρ corresponding to round r, whereas $timeout(r)$ triggers when the timeout is completed and $clearTimeout(r)$ removes any active timeout corresponding to round r. The function $time()$ returns the local time of the party and when input with a round r, $time(r)$ returns the local time when the party adopted round r. In the case that the party never adopted round r, $time(r)$ returns the time when the party adopted the smallest round $r' > r$. The function $RR(r)$ takes a round as an input and returns the party by the round-robin mechanism in round r, a block from this party is also referred to as *anchor*. In case multiple blocks exist, e.g. due to Byzantine behavior, the function returns both blocks. The variable $DAG(r)$ returns the set of blocks from round r in the local view of the party. The operation $maxAnchor(\mathcal{A})$ takes a set of blocks $\mathcal{A}$ as an input and returns the anchor from the highest round. Given a round r, the function $quorum(r)$ returns 1 if there are blocks from at least $n - f$ parties in $DAG(r)$. The function $anchor(r)$ returns 1 if there is a block created by the anchor of r in $DAG(r)$, and $suppAnchor(r)$ returns 1 if there is an anchor block in r that is supported by at least $n - f$ parties in $DAG(r + 1)$.

5 Analysis

Black Marlin (Algorithm 2) is designed for partially synchronous networks, which means that it may not be alive during the asynchronous period. However, its safety properties are still guaranteed during this period. This fact is reflected in the analysis below.

5.1 Safety

The results in this section apply to every block, even during the asynchronous phase.

▶ **Lemma 2.** *Every honest message* $[\text{BLOCK}, B', \mathcal{H}', j, r]$ *received by an honest party* i *is valid, i.e., satisfies line L58 in the view of* i*.*

Proof. A message $[\text{BLOCK}, B', \mathcal{H}', j, r]$ is valid if block B' and history $\mathcal{H}'$ satisfy the validity predicate V. If the sender is honest, both B' and $\mathcal{H}'$ satisfy the validity predicate. Thus, it is only left to prove that party i has enough information to verify the validity predicate. The construction of the history $\mathcal{H}'$ guarantees this fact. ◀

Lemma 2 applies whenever an honest party sends a message. For the sake of readability, we omit explicit references to this lemma each time an honest message is sent.

▶ **Lemma 3.** *Given two honest parties* i *and* j *and blocks* B_i *and* B_j *from round* r*, if* B_i *satisfies line L16 in the view of party* i *and* B_j *satisfies line L16 in the view of party* j*, then* $B_i = B_j$*.*

Proof. Given a block B_i created by party k in round r that satisfies line L16 in the view of some party, it follows that $creator(B_i) = RR(r)$ and $supp(B_i) \geq n - f$. Recall that, by definition of the support function, an honest party supports at most one block per party per round. Therefore, if blocks B_i and B_j both satisfy line L16 in the views of parties i and j respectively, then the following condition must be satisfied $creator(B_i) = creator(B_j) = RR(r)$ and $supp(B_i), supp(B_j) \geq n - f$. Given the corruption threshold condition of $n \geq 3f + 1$, it is impossible for two distinct blocks to each have support from at least $n - f$ parties. Hence, it must be the case that $B_i = B_j$. ◀

Lemma 3 states that at most one block is honestly committed per round. Furthermore, these committed blocks create a chain in the DAG, as we prove in the following lemmas.

Algorithm 2 Black Marlin (party i).

```
33: upon init do
34:     ab-broadcast(B, i, 1)
35:     send message [BLOCK, B, {}, i, r] to every party
36:     r ← 1
37:     setTimeout(1, 2Δ)

38: upon ∃r' ≥ r : quorum(r') do
39:     setTimeout(r', 2Δ)

40: upon ∃r' ≥ r : quorum(r') ∧
        (anchor(r') ∧ suppAnchor(r' − 1) ∧ suppAnchor(r' − 2)) ∨ timeout(r')) do
41:     clearTimeout(r')
42:     delivery(r')
43:     refs ← ∅
44:     r_ℓ ← min{r' ∈ ℕ : |time() − time(r)| ≤ 3Δ}
45:     weak ← ⋃_{r*=r_ℓ}^{r'−1} DAG(r*)
46:     for j ∈ parties(DAG(r)) do
47:         B' ← B ∈ DAG(r) : creator(B') = j        if multiple B', select any
48:         refs ← refs ∪ H(B')
49:         weak ← weak \ past(B')
50:     refs' ← ∪_{B*∈weak} H(B*)
51:     r ← r' + 1
52:     B ← [r, i, refs, refs']
53:     ab-broadcast(B, i, r)
54:     for j ∈ 𝒩 do
55:         send message [BLOCK, B, DAG \ history[j], i, r] to party j
56:         history[j] ← DAG

57: upon receiving message [BLOCK, B', 𝓗', j, r'] do
58:     if V(B') ∧ V(𝓗') then
59:         DAG ← DAG ∪ {B'} ∪ 𝓗'
60:         history[j] ← history[j] ∪ 𝓗' ∪ {B'}
```

▶ **Lemma 4.** *Consider an honest party i and the maximal round r such that $DAG(r) \neq \emptyset$ in the view of party i, then $|DAG(r')| \geq n - f$ for every round $r' < r$.*

Proof. Consider the maximal round r and a block $B \in DAG(r)$. Since B is a valid block, there exists valid blocks from at least $n - f$ parties in $strong(B)$ corresponding to round $r - 1$. Thus, $|DAG(r - 1)| \geq n - f$. Now, consider any block in $DAG(r - 1)$ and iterate recursively. We conclude that $|DAG(r')| \geq n - f$ for every round $r' < r$. ◀

In contrast to traditional leader-based protocols, a block in DAG-based protocols must be in the past of an overwhelming majority of blocks to ensure its delivery. The following result formalizes this requirement for committed blocks.

▶ **Lemma 5.** *If a block B from round r satisfies line L16 in the view of some honest party i, then every valid block B' from round $r' \geq r + 2$ satisfies $B \in past(B')$.*

Proof. By definition of line L16, $supp(B) \geq n - f$, thus $B \in past(B')$ of blocks B' from at least $n - f$ different parties in round $r + 1$. We proceed by induction over $\rho := r' - r$:

- Let B' be any block from round $r + 2$. By construction, there are blocks from at least $n - f$ in $DAG(r + 1)$, and up to f of them could be Byzantine. Block B is in the past of blocks from round $r + 1$ from at least $n - f$ different parties, thus $B \in past(B')$. We conclude $B \in past(B')$ of any block B' from round $r + 2$.
- Assume that every block B_0 from round $r + \rho_0$ satisfies $B \in past(B_0)$ and let B' be a block from round $r' = r + \rho_0 + 1$. We have that $B \in past(B')$, since there are at least $n - f$ blocks in $past(B')$ and every block B_0 from round $r + \rho_0$ fulfills $B \in past(B_0)$.

We conclude that any block B from round r that satisfies line L16 in the view of some honest party, then every block B' from round $r' \geq r + 2$ satisfies $B \in past(B')$. ◀

▶ **Lemma 6.** *If honest parties i and j commit blocks B_i and B_j respectively, then $B_i = B_j$, $B_i \in past(B_j)$, or $B_j \in past(B_i)$.*

Proof. Let $round(B_i)$ and $round(B_j)$ denote the rounds in which parties i and j commit blocks B_i and B_j, respectively. We analyze three cases based on the values of these rounds:

- **Case 1:** $round(B_i) = round(B_j)$.
  By Lemma 3, at most one block can be committed per round. Hence, $B_i = B_j$.
- **Case 2:** $round(B_i) < round(B_j)$. We further distinguish two sub-cases:
  - **Case 2a:** $round(B_j) = round(B_i) + 1$.
    By the second condition in line L16, there exists a block $B'_j \in DAG(round(B_j))$ such that $creator(B_j) = creator(B'_j)$ and $B_i \in strong(B'_j)$. Since Byzantine parties may create multiple blocks, we cannot immediately conclude that $B_j = B'_j$. However, the third condition of line L16 ensures that both B_j and B'_j satisfy $supp(B_j), supp(B'_j) \geq n - f$, which, by Lemma 3, implies $B_j = B'_j$. Thus, $B_i \in past(B_j)$.
  - **Case 2b:** $round(B_i) + 1 < round(B_j)$.
    In this case, Lemma 5 guarantees that $B_i \in past(B_j)$.
- **Case 3:** $round(B_i) > round(B_j)$.
  This is symmetric to Case 2 and follows by applying the same reasoning, yielding $B_j \in past(B_i)$.

We conclude that any two blocks committed by honest parties satisfy that both parties commit the same block or blocks in the past of each other. ◀

This chain of committed blocks is common across honest parties, thus defining a partial order common to every honest party.

5.2 Liveness

Liveness is guaranteed only after the global stabilization time. For this reason, the majority of the results presented here apply only to blocks that are *ab-broadcast* after time GST.

▶ **Lemma 7.** *If an honest party invokes ab-broadcast(B, i, r) at time $T \geq GST$, then every honest party j that invokes ab-broadcast(B', j, r), does it before time $T + 3\Delta$.*

Proof. Assume that party i invokes *ab-broadcast*(B, i, r) at time $T \geq GST$. Then, according to line L55, i sends a message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ to every party. An honest party j receives this message by time $T + \Delta$. Since the message includes all the

strong references of B, party j's view satisfies $|DAG(r-1)| \geq n-f$. As a result, j can create a block for round r once the anchors have sufficient support or a timeout of duration 2Δ triggers. Therefore, j can create its block by time $T+3\Delta$. ◀

Lemma 5 guarantees that committed anchor blocks are in the past of future blocks. The result below generalizes this property to all honest blocks created after the global stabilization time GST.

▶ **Lemma 8.** *If honest party i invokes ab-broadcast(B,i,r) at time $T \geq GST$, then $B \in past(B')$ of every honestly ab-broadcast(B') block B' after time $T+\Delta$*

Proof. Assume that party i invokes *ab-broadcast*(B,i,r) at time T, then i sends a message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ to every party (L55). Any block B' created by party j after receiving the message $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ satisfies $B \in past(B')$. Lemma 7 guarantees that the message $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$ is not garbage collected, thus proving the statement. ◀

▶ **Lemma 9.** *Given an honest party i and a round $r \in \mathbb{N}$, $\exists r' \geq r$ such that i invokes ab-broadcast(B,i,r') for some block B.*

Proof. Let r be a round, and define $r' \geq r$ as the first round such that no honest party has executed it before time GST, and $RR(r') = i$. We show that party i eventually invokes *ab-broadcast*(B,i,r') for some block B.

Consider line L40 as executed in round r' by any honest party j. Since i is the leader, no honest party concludes round r without a block from i unless its timeout triggers. Consider j to be the first honest party that concluded round $r'-1$, the duration of its timeout is 2Δ. This guarantees that party i receives the quorum of round $r-1$ sent by j (at most Δ delay) before any honest party concludes round r', thus party i invokes ab-broadcast(B,i,r') ◀

▶ **Lemma 10.** *At time $T \geq GST$, if an honest party concludes a round r (L40) such that the anchors from rounds $r-2$ and $r-1$ are honest, then $suppAnchor(r-2) \geq n-f$ and $suppAnchor(r-1) \geq n-f$.*

Proof. At time $T \geq GST$, assume an honest party concludes round r (L40) and that the anchors from rounds $r-2$ and $r-1$ are also honest. By construction, if a party creates a block for round r at time T', then every honest party receives this block by time $T'+\Delta$ and subsequently creates its own block, which is received by all honest parties by time $T'+2\Delta$.

Given that both anchors from rounds $r-2$ and $r-1$ are honest, it follows that $suppAnchor(r-2) \geq n-f$ and $suppAnchor(r-1) \geq n-f$, since there are at least $n-f$ honest parties contributing to support anchors in each round. ◀

Lemma 10 states that after the GST, the timeout is not triggered with honest anchors.

▶ **Lemma 11.** *After the global stabilization time, given a round r, the probability that honest party i commits a block in r is at least $\frac{(n-t)^2}{n^2}$. Thus the expected number of rounds until a block is committed is at most $\frac{n^2}{(n-t)^2}$.*

Proof. Given a round r, party i commits a block $B \in DAG(r-2)$ (L16) if and only if the following conditions are satisfied:

- The creator of B has been elected by the round-robin mechanism in round $r-2$, i.e., $creator(B) = RR(r-2)$, and the block has sufficient support, $supp(B) \geq n-f$.
- There exists a block $B' \in DAG(r-1)$ such that $B \in strong(B')$, $creator(B') = RR(r-1)$, and $supp(B') \geq n-f$.

Assume that the anchors from rounds $r-2$ and $r-1$ are both honest; this implies the existence of $B \in DAG(r-2)$ and $B' \in DAG(r-1)$, as described above. According to Lemma 10, both blocks have enough support: $supp(B) \geq n-f$ and $supp(B') \geq n-f$. Since there are $n-t$ honest parties, the anchors from rounds $r-2$ and $r-1$ are both honest with probability $\frac{(n-t)^2}{n^2}$. Hence, party i commits a block in round r with probability at least $\frac{(n-t)^2}{n^2}$ and commits every a block every at most $\frac{n^2}{(n-t)^2}$ rounds on average. ◀

This lemma shows that, under maximum number of corruptions, honest parties commit a block every at most 9/4 rounds. Nonetheless, a clever adversary may delay commitment for a short period of time (t rounds), at the cost of allowing the network to commit optimally (every round) outside this period.

▶ **Lemma 12.** *Given two honest parties i and j and a block B, then i commits B if and only if j eventually commits B.*

Proof. Assume that an honest party i commits a block B. By Lemma 11, any other honest party j will eventually commit a block B' such that $round(B') \geq round(B)$. Lemma 6 applied to blocks B and B' implies that either $B = B'$ or $B \in past(B')$. Since $round(B') \geq round(B)$, it cannot be the case that $B' \in past(B)$ unless $B = B'$. If $B' = B$, then party j directly commits the same block. If $B \in past(B')$ and $B \neq B'$, then by construction of the delivery function, party j must have also committed B. Therefore, we conclude that if an honest party i commits a block B, then every honest party j eventually commits B. ◀

5.3 Main Result

Previous lemmas are sufficient to show that Black Marlin implements atomic broadcast.

▶ **Theorem 13.** *Black Marlin (Algorithm 2) implements atomic broadcast.*

Proof. We proceed property by property:

Validity. Assume an honest party i invokes $ab\text{-}broadcast(B, i, r)$ after the global stabilization time GST. According to the delivery function (L14), party i outputs $ab\text{-}deliver(B, i, r)$ the first time it observes a block B' such that $B \in past(B')$ and B' satisfies line L16. Lemma 8 guarantees that B is eventually in the past of every valid block. Lemmas 9 and 11 ensure that B' is eventually committed, hence party i outputs $ab\text{-}deliver(B, i, r)$.

Agreement. Suppose an honest party i outputs $ab\text{-}deliver(B, j, r)$. By Algorithm 2, this means a block B^* with $B \in past(B^*)$ satisfies line L16. For any other honest party j, Lemmas 9 and 11 guarantee that j eventually commits a block B' with $B^* \in past(B')$, and thus $B \in past(B')$. Therefore, party j eventually $ab\text{-}delivers(B, j, r)$, ensuring agreement.

Integrity. The delivery function (L16) ensures that honest party j keeps track of all previously delivered blocks. Consequently, j outputs $ab\text{-}deliver(B, i, r)$ at most once per party i and round r. Furthermore, if i is honest, j outputs $ab\text{-}deliver(B, i, r)$ only if i previously invoked $ab\text{-}broadcast(B, i, r)$ as Byzantine parties cannot impersonate honest parties.

Total order. By definition of the delivery function, the order of delivery of non-anchor blocks is determined by the order in which the anchor blocks are committed (L16). Thus, it suffices to show that honest parties commit anchors in the same order. Lemma 12, guarantees that every honest party commits the same set of blocks. The delivery function called with block B guarantees that any anchorin the past of B is committed before B, Lemma 6 guarantees that any committed blocks (possibly by different parties) are in the past of each other. Thus, honest parties commit anchors in the same order. ◀

5.4 Communication and Time Complexity

Previously, we showed that Algorithm 2 is both safe and live in a partial synchronous network. In particular, we demonstrated that blocks are eventually delivered, avoiding any concrete analysis of the communication and time complexity of the algorithm.

5.4.1 Communication complexity

Each round, a party sends at most one message of the form $[\text{BLOCK}, B, DAG \setminus history[j], i, r]$. We assume that the round number can be expressed in logarithmic size $\mathcal{O}(\ln(n))$ (note that the round number grows more slowly than the number of blocks). The identifier of the party can also be expressed in $\mathcal{O}(\ln(n))$ number of bits. For the computation of the size of $DAG \setminus history[j]$ we assume that the Byzantine parties do not send multiple equivocating blocks, as these blocks include signatures that implicate the party, thus the size of $DAG \setminus history[j] \leq \mathcal{O}(n)$. We conclude that the amount of bits sent per honest party and round is $\mathcal{O}(n(|B| + n))$, hence packing n transactions of constant size in B we obtain a communication complexity of $\mathcal{O}(n^2)$ with an amortized complexity of $\mathcal{O}(n)$, both optimal [1].

5.4.2 Time complexity

Lemma 11 gives an upper bound on the expected number of rounds of communication between anchor blocks are committed. An anchor block is committed on average every $\frac{n^2}{(n-t)^2}$ rounds in the presence of t corruptions: in the worst case, every 9/4 round of communication, and every round of communication in the good case. Note that in round r, an anchor from round $r-2$ is committed. Considering latency to be the number of rounds of communication since an anchor block is broadcast until it is delivered we obtain 4.25 and 3 rounds of communication in the average and good case, respectively. It is important to remark that in contrast to most DAG-based protocols, we are measuring in rounds of communication and not instances of reliable broadcast. Thus, Black Marlin not only achieves $\mathcal{O}(1)$ time complexity in expectation, but also the best concrete time complexity concerning DAG-based protocols [24, 25, 5, 15, 6], as their use of reliable broadcast hinders the protocols.

6 Benchmarking

We evaluate the performance of our implementation with respect to the metrics of throughput and latency for a varying number of parties and network delays.

6.1 Experiments

We conducted all benchmarks locally on a MacBook Pro equipped with an M1 Pro chip, 16 GB of RAM, 10 cores (8 performance and 2 efficiency), and running macOS Sequoia Version 15.0.1. To ensure a fair comparison, we modified the benchmarking framework developed for Bullshark [26, 21][2] to incorporate our implementation of Black Marlin. The biggest difference between Black Marlin and Bullshark is the use of common coins by Bullshark. However, their implementation also uses round-robin[3]. This approach allows a close performance

[2] Bullshark has been chosen over more recent frameworks such as Mysticeti [18] due to its closer similarity to Black Marlin and the timing of this project.

[3] `https://github.com/facebookresearch/narwhal/blob/bullshark/consensus/src/lib.rs` L202.

comparison between Black Marlin and Bullshark and also demonstrates that Bullshark's preprocessing techniques can be adapted to Black Marlin. Note that the actual performance of Bullshark is worse due to the lack of common coins in the implementation.

To emulate realistic network conditions despite running benchmarks on a single machine, we modified both the Black Marlin and Bullshark implementations to introduce configurable network delays. Specifically, messages are delivered based on a Poisson distribution with the expected value $\Delta/2$, where Δ represents the maximum network delay assumed by the protocol. This design ensures that with high probability, the actual message delay remains below Δ. For instance, when $\Delta = 400$ ms, the average delivery time for a message is 200 ms.

For each experimental run, we measure *throughput* as the total number of transactions delivered divided by the total duration of the run. The latency of a transaction *tx* is defined as the time elapsed from the creation of the block containing *tx* until it is *ab-delivered*. The *latency* for a run is the average latency of all *ab-delivered* transactions in the run. Every party behaves honestly during the benchmarking.

Each data point in Figure 3 represents the average throughput or latency over 10 independent runs, each lasting 60 seconds. Error bars indicate two standard deviations. We report results across varying numbers of parties (4, 10, and 13) and network delays ($\Delta = 200$, 400, 600, 800, and 1000 ms). All other parameters remained constant throughout the experiments: one worker per party, a transaction size of 512 bytes, and a transaction injection rate of 50,000 transactions per second.

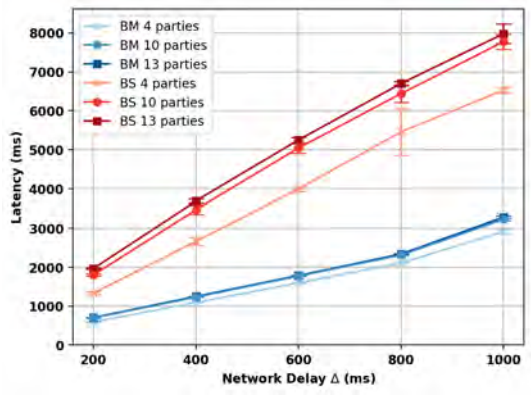
6.2 Results

The results of the experiments described above are summarized in Figure 3, Black Marlin is represented by blue tones and Bullshark by red ones. The two figures of merit that we consider are throughput and latency.

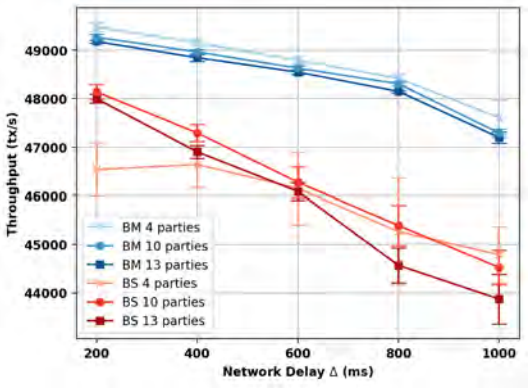
Black Marlin consistently achieves lower latency than Bullshark across all configurations. The latency improvement ranges from a factor of 2 to 3, depending on the network delay. For example, with 10 parties and a network delay of $\Delta = 200$ ms, Black Marlin achieves a latency of 696 ± 5 ms, compared to 1800 ± 54 ms for Bullshark. At $\Delta = 1000$ ms, Black Marlin's latency is 3226 ± 55 ms, while Bullshark's latency increases to 7773 ± 360 ms. This improvement is expected: Black Marlin eliminates the need for reliable broadcast, reducing latency by approximately a factor of 2. Additionally, its more frequent anchor selection contributes to a constant-factor latency reduction, further widening the latency gap.

Black Marlin also achieves consistently higher throughput than Bullshark across all network delay scenarios. In contrast to the more pronounced latency gains, the throughput improvement is by a constant margin. This behavior can be explained by the way Bullshark constructs blocks: a party running Bullshark accumulates batches of transactions from workers until it is ready to broadcast a new block. As network delay increases, blocks become larger, which helps mitigate throughput loss. Therefore, the $\sim 2\times$ latency improvement observed with Black Marlin does not directly translate into an equivalent gain in throughput – unless the number of workers or the network delay is significantly increased. However, these adjustments were beyond the scope of this study. We were unable to increase the number of workers due to hardware limitations, and we excluded network delays greater than 1 second, as such conditions are considered unrealistic. This block size phenomenon also explains the erratic throughput results observed for Bullshark under configurations with 4 parties and low network delays. In these cases, small block sizes limited Bullshark's throughput.

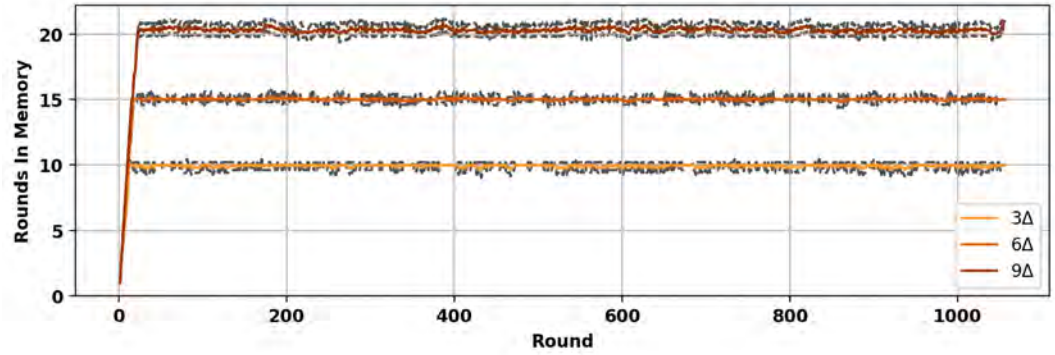
Figure 3 also demonstrates that Black Marlin's garbage collection mechanism is robust to overestimations of the actual network delay. Specifically, a constant-factor overestimation leads to only a constant increase in the number of rounds that parties must retain in memory. This implies that even for arbitrarily long executions, Black Marlin only requires $\mathcal{O}(n)$ memory, provided the network delay is overestimated by a constant factor.



(a) Latency of Black Marlin and Bullshark.



(b) Throughput of Black Marlin and Bullshark.



(c) Garbage collection as a function of the estimated network delay.

Figure 3 Black Marlin is shown in blue, Bullshark in red. Darker tones indicate higher numbers of parties: $n = 4$ (cross), $n = 10$ (circle), $n = 13$ (square). Black Marlin outperforms Bullshark in latency and throughput: latency improves by a factor of 2–3; throughput sees a smaller but consistent gain. Bullshark's block creation strategy explains this modest throughput gap – it produces larger blocks to compensate for delays, up to a size limit. This also accounts for Bullshark's irregular throughput at $n = 4$ under low delay, where small blocks constrain performance. Plot (c) shows the number of rounds stored in memory by Black Marlin as a function of estimated network delay. The actual delay is $\Delta = 600$ ms, modeled as a Poisson distribution with mean 300 ms. We vary the garbage collection threshold from 3Δ to 6Δ and 9Δ, observing linear growth in retained rounds. As each round contains $\mathcal{O}(n)$ blocks, memory usage remains $\mathcal{O}(n)$.

Overall, Black Marlin outperforms Bullshark in our implementation, which uses the same codebase and experimental conditions for both protocols. Unfortunately, source code for more recent DAG-based protocols is not publicly available, preventing a direct empirical comparison. However, as we demonstrate in Section 5.4, Black Marlin also surpasses these protocols at a theoretical level in terms of communication and latency complexities while allowing implementations with $\mathcal{O}(n)$ memory.

References

1 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. In *PODC*, pages 331–341. ACM, 2021. doi:10.1145/3465084.3467899.

2 Ignacio Amores-Sesar and Christian Cachin. We will DAG you. In *ESORICS Workshops (1)*, volume 15263 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2024. doi:10.1007/978-3-031-82349-7_19.

3 Ignacio Amores-Sesar, Christian Cachin, and Philipp Schneider. An analysis of avalanche consensus. In *SIROCCO*, volume 14662 of *Lecture Notes in Computer Science*, pages 27–44. Springer, 2024. doi:10.1007/978-3-031-60603-8_2.

4 Ignacio Amores-Sesar, Christian Cachin, and Enrico Tedeschi. When is spring coming? A security analysis of avalanche consensus. In *OPODIS*, volume 253 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.OPODIS.2022.10.

5 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast and robust! In *NSDI*, pages 813–826. USENIX Association, 2025. URL: https://www.usenix.org/conference/nsdi25/presentation/arun.

6 Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified dags. In *NDSS*. The Internet Society, 2025. URL: https://www.ndss-symposium.org/ndss-paper/mysticeti-reaching-the-latency-limits-with-uncertified-dags/.

7 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985. doi:10.1145/4221.214134.

8 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

9 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51. ACM, 1993. doi:10.1145/167088.167105.

10 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

11 Andrew Cullen, Pietro Ferraro, Christopher K. King, and Robert Shorten. On the resilience of dag-based distributed ledgers in iot applications. *IEEE Internet Things J.*, 7(8):7112–7122, 2020. doi:10.1109/JIOT.2020.2983401.

12 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

13 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

14 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

15 Neil Giridharan, Florian Suri-Payer, Ittai Abraham, Lorenzo Alvisi, and Natacha Crooks. Autobahn: Seamless high speed BFT. In *SOSP*, pages 1–23. ACM, 2024. doi:10.1145/3694715.3695942.

16 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

17 Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *DISC*, volume 281 of *LIPIcs*, pages 26:1–26:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.26.

18 Mysten Labs. Github, accessed on April 2025. URL: https://github.com/asonnino/mysticeti/tree/paper.

19 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998. doi:10.1145/279227.279229.

20 Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the tangle. *Comput. Ind. Eng.*, 136:160–172, 2019. doi:10.1016/J.CIE.2019.07.025.

21 Facebook Research. Bullshark implementation. Github, accessed on April 2025. https://github.com/facebookresearch/narwhal/tree/bullshark.

22 Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR*, abs/1906.08936, 2019. arXiv:1906.08936.

23 Victor Shoup. Blue fish, red fish, live fish, dead fish. *IACR Cryptol. ePrint Arch.*, page 1235, 2024. URL: https://eprint.iacr.org/2024/1235.

24 Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving latency of dag-based BFT. *IACR Cryptol. ePrint Arch.*, page 472, 2024. URL: https://eprint.iacr.org/2024/472.

25 Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024. doi:10.1007/978-3-031-78676-1_6.

26 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *CCS*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

27 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

Hierarchical Consensus: Scalability Through Optimism and Weak Liveness

Pedro Antonino
The Blockhouse Technology Ltd., Oxford, UK

Antoine Durand
The Blockhouse Technology Ltd., Oxford, UK

A. W. Roscoe
The Blockhouse Technology Ltd., Oxford, UK
University College Oxford Blockchain Research Centre, UK

Abstract

Scalability is a central concern of Byzantine Fault Tolerant (BFT) distributed protocols. The ubiquitous approach to work around the well-known Dolev-Reischuk $\Omega(n^2)$ communication complexity lower bound is to use a random selection process to draw a hopefully small committee from a population of agents to run the communication-heavy protocol. We propose a notion of *hierarchical consensus* that combines two sub-protocols: an optimistic primary sub-protocol that can tolerate less than 1/2 failures and a fallback secondary protocol that can tolerate less than 1/3 failures; we achieve the higher failure threshold by requiring a weaker notion of liveness for the primary. This distinction between the level of fault tolerance between primary and secondary is reflected in the size of committees implementing these protocols. For a population of agents with close to 2/3 of honest agents, we need to select a committee with hundreds of agents to reach the level of tolerance expected for the primary, whereas we need thousands to reach the level expected for the secondary with a very small probability of error ϵ. Our hierarchical construct is such that if the primary comes to a decision, it can simply propagate it to the secondary protocol, so it does not need to properly engage in an agreement protocol independently. Our architecture is flexible and allows us to use our technique for most protocols that are based on random sampling. By studying hierarchical protocols, we discovered new theoretical results of independent interest. Specifically, the ability to handover from a primary protocol requires a new *Justifiability* property that allows agents to pre-decide on a value, such that if the protocol decides, it must be on that pre-decided value.

2012 ACM Subject Classification Networks → Network protocol design; Theory of computation → Distributed algorithms

Keywords and phrases Hierarchical, Handover, Justifiability, Consensus, Distributed Systems, Blockchain

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.6

1 Introduction

The scalability of Byzantine fault-tolerant (BFT) protocols [20, 6, 14] has been the subject of renewed attention since the advent of large-scale distributed applications such as blockchains [23, 27, 19]. Permissionless protocols allow in principle an unbounded number participants, so their communication cost (i.e. the number of bits sent by honest agents) must not overwhelm the network.

Deterministic BFT consensus protocols are subject to the well-known Dolev-Reishuk $\Omega(n^2)$ lower bound on communication cost [18, 8], where n is the number of agents participating in the protocol. Scalable agreement protocols try to work around this lower bound by randomly selecting committees: members are picked randomly and independently from an underlying population of agents with a defined proportion of honest-to-Byzantine agents. The goal of this process is to minimise the size n of the committee selected: find the smallest n such


© Pedro Antonino, Antoine Durand, and A. W. Roscoe;
licensed under Creative Commons License CC-BY 4.0



that the committee has the expected ratio of honest-to-Byzantine agents. Of course, by minimising n, the communication cost $\Omega(n^2)$ is reduced. However, selecting a committee size that ensures *with high probability* that more than 2/3 agents are honest – the lower bound for partially synchronous protocols – from a population of agents with a similar ratio of honest agents leads to committee sizes in the thousands, hindering the end goal of scalability.

Our work stems from the observation that it is possible to have much smaller committees if we select, instead, committees with more than 1/2 honest agents in a population with roughly a 2/3 probability of an agent being honest. We rely on the observation from Momose and Ren [22] that BFT protocols can set different fault thresholds t_{safe} and t_{live} for their safety and liveness properties, respectively; protocols that do not use this multi-threshold approach implicitly make $t_{safe} = t_{live}$. The committee size can be set to guarantee fewer than t_{safe} faults, while keeping a reasonable probability that there are fewer than t_{live} faults as well. Since t_{safe} can be set to a value that is higher than t_{live}, the committee can be much smaller. For instance, we can have that $t_{safe} < n/2$ whereas for traditional (single-threshold) protocols $t_{safe} = t_{live} < n/3$.

We propose the notion of *hierarchical consensus* as a framework to create efficient BFT consensus protocols by exploring this threshold flexibility. Although the concept of a hierarchical architecture itself is not novel, it presents an extension of the traditional notion of agreement protocols in the sense that it allows some protocol agents to output values that are *not decisions*. Intuitively speaking, this concept combines two sub-protocols: an optimistic primary protocol that is safe and only weakly live, and a fallback secondary protocol that is safe and live. By only requiring weak liveness in non-optimistic executions, the secondary can be guaranteed to execute when necessary. Thus, in optimistic executions of this protocol, the primary sub-protocol comes to a decision that is simply propagated to the secondary protocol; the latter does not have to be active at all. It is important to note here that t_{safe} also ensures weak liveness for the primary; this notion allows the fallback to be started when the primary agents do not reach a decision.

We introduce the notion of a *Justifiable reliable broadcast* (JRBC) protocol as a way to characterise protocols that can be used as a primary. It captures the safety and weak liveness requirements that enable a safe handover to the secondary sub-protocol in the case that the primary is unable to decide. We show that for a JRBC protocol, it must be that $2t_{safe} + t_{live} < n$, whereas non-Justifiable protocols are only bound by the strictly weaker relation $t_{safe} + 2t_{live} < n$. This impossibility shows for the first time that there are extra constraints when implementing such primary/secondary optimistic architecture. We also propose an implementation *psync JRBC* of a partially synchronous JRBC that demonstrates that this bound is tight. Notably, a similar multi-threshold approach to consensus has been tried [11], but that attempt falls short of reaching the same performance gains, precisely because the Justifiability requirements that we identified were not met. In [25, 26], the authors propose handover protocols that are similar in nature to ours but different in design: they rely on a different handover mechanism and communication model.

We propose the *handover construction* $\mathcal{P}^H$ that creates a hierarchical consensus protocol by combining a Justifiable RBC protocol $\mathcal{P}_o$ (with participant agents $\mathcal{A}_o$) and a consensus protocol $\mathcal{P}_l$ (with participant agents $\mathcal{A}_l$). Our *psync JRBC* implementation can be combined with a consensus implementation of choice using this construction to create a hierarchical consensus implementation. Our *psync JRBC* implementation shows that JRBC protocols can tolerate up to $\lfloor \frac{n-1}{2} \rfloor$ faults while remaining safe and weakly live as opposed to the $\lfloor \frac{n-1}{3} \rfloor$ failure tolerance level for live consensus protocols; this difference in their ability to tolerate failures is crucial to achieve efficiency. In an optimistic run, the agents implementing $\mathcal{P}_o$ can

come to a decision independently of the agents in $\mathcal{P}_l$. Thus, by minimising the size of $\mathcal{A}_o$, we can improve $\mathcal{P}_o$'s communication complexity. As an additional bonus, although optimistic executions can happen with up to t_{live} faults, the probability of having an optimistic execution only depends on *the actual* number of faults. Of course, non-optimistic runs can be inefficient as both sub-protocols attempt to reach agreements in succession.

We employ a notion of *stochastic reasoning* to demonstrate the impact that these levels of tolerance have on the size of committees – that is, $\mathcal{A}_o$ and $\mathcal{A}_l$ – necessary to implement $\mathcal{P}_o$ and $\mathcal{P}_l$. For an error parameter ϵ, a probability p of drawing an honest agent in a population of agents, and a threshold t of Byzantine agents, we can calculate the smallest size n such that for a collection of n agents randomly and independently drawn from this population of agents the probability that this collection has more than t is at most ϵ. Thus, by choosing ϵ exponentially small in the security parameter, the probability that this collection does not respect the tolerance level t is negligible. As a concrete example, for $p = 0.68$ and $\epsilon = 10^{-18}$, $t = \lfloor \frac{n-1}{2} \rfloor$ can be achieved with committee size $n = 543$ whereas $t = \lfloor \frac{n-1}{3} \rfloor$ is achieved with $n = 94366$. This demonstrates the significant efficiency gains that can be achieved by our framework. In the context of blockchains, we can implement (financial) incentive structures to encourage even Byzantine agents to participate in the decision of $\mathcal{P}_o$. Note that Byzantine agents in $\mathcal{A}_o$ can prevent $\mathcal{P}_o$ from reaching a decision, but they cannot prevent $\mathcal{P}^H$ from reaching one. Thus, the incentive structure must be designed to make delaying a decision (from $\mathcal{P}_o$ to $\mathcal{P}_l$) not worthwhile.

We summarise our contributions below:

- A *hierarchical consensus* framework that can be used to design efficient agreement protocols by combining a primary (tolerating $< 1/2$ failures) and a secondary (tolerating $< 1/3$ failures) protocol, the former of which does not need to be strictly live.
- A characterisation of primary protocols via the notion of *Justifiable* protocols and an impossibility result that demonstrates some notion of Justifiability is necessary for hierarchical consensus. We additionally show that partially synchronous JRBC protocols must respect $2t_{safe} + t_{live} < n$, and we build the implementation *psync JRBC* that also demonstrates the tightness of this bound.
- A *handover construction* that demonstrates how a JRBC protocol and consensus protocol implementations can be combined to implement a hierarchical consensus protocol.
- A *stochastic analysis* of (randomly sampled) committee sizes for typical primary and secondary protocols that demonstrates the level of efficiency gains (in terms of communication complexity) that can be realised by selecting smaller committees for the primary.

Paper organisation. Section 2 reviews the research literature that is closely related to our paper. In Section 3, we present the necessary background to make the paper self-contained. Section 4 presents our framework for efficient BFT agreement using hierarchical consensus in detail. In Section 5 we show how to instantiate that framework. Finally, we present our concluding remarks in Section 6.

2 Related Work

De la Rocha et al. [12] have proposed a hierarchical architecture for consensus protocols. They focus on running independent, arbitrary consensus instances where the hierarchical relation concerns the instances' trust assumptions. So, their technical contributions are vastly different from ours.

Similar techniques. The work from Bernardo et al. [11] is closest to ours. They also split the tolerance thresholds into t_{safe} and t_{live} to sample committees (called shards) of smaller size, with the same resulting probability of being live. They have a control shard which is large enough to always guarantee liveness and safety, like our secondary protocol. The main difference with our work is that they do not require Justifiability (or anything equivalent), and enjoy the weaker, common restriction of $t_{safe} + 2t_{live} < n$ instead of $2t_{safe} + t_{live} < n$. However, their solution is thereby bound by the impossibility in Theorem 9, and they cannot build a handover-like mechanism. Instead, the output of the smaller shards must always be fed to the control shard, meaning that the large committee runs an agreement protocol *for every finality decision*, even when the optimistic protocol is live. This related work illustrates well how our Justifiability notion is critical to achieving the efficiency benefits of multi-threshold BFT protocols.

In [25, 26], the authors propose handover protocols that are similar in intent to ours but different in design. For instance, while we have the primary triggering the handover, in those papers, the secondary does that, usually via a timeout. Even though the secondary triggers the handover, their system still enjoys the same optimistic behavioural pattern whereby the primary can come to a decision at which point the secondary protocol does not need to run. Also, the communication model employed there is different to ours: while they make use of signals (i.e. single-writer multiple-readers shared memory locations for storing signals), we use point-to-point communication.

Optimistic efficiency. More generally, our work aims to offer a lower communication cost under favourable executions, which falls into the realm of *optimistically efficient* protocols, also known as protocols with adaptive communication cost. Although such protocols have received less interest than optimistically low-latency (e.g., responsive) protocols, there are notable contributions.

The protocol from Gelles and Komargodski [17] has an overall communication cost that is constant in fully honest executions, and a (balanced) $O(n^{\frac{3}{2}})$ otherwise. On the other hand, their protocol is in the unauthenticated, strongly synchronous setting, whereas ours is authenticated, partially synchronous. Cohen et al. [10] proposed a $O(nf)$ algorithm where f is the *actual* number of faults, using threshold signatures. This was subsequently improved upon by Civit et al. [7] and Elsheimy et al. [15] to different variants of consensus and to extend the efficiency gains to the input size. Their protocols are synchronous and tolerate up to a $\frac{1}{2}$ fraction of Byzantine agents. Interestingly, except for our work, there is no optimistically efficient partially synchronous protocol that we know of. In addition, one appeal of our technique is that it can be applied to any committee-sampling protocol, which is orthogonal to the technique employed by other optimistically efficient protocols.

Justifiability concepts. Interestingly, although our Justifiability notion is new, similar concepts can be found in somewhat unrelated parts of the literature. We expect that fully relating those concepts could lead to deeper insights. Abraham et al. [1] proposed a *Binding* property which is almost the same as the existence of a pre-decision, except that it is in the context of Crusader Agreement. Deligios et al. [13] proposed a similar primary/fallback architecture, but dually to our work, the fault thresholds are split depending on network synchronicity instead of liveness/safety properties. They show analogous feasibility and impossibility results for their synchronous/asynchronous fallback architecture. More generally, there are a few other works in other contexts that explore the trade-offs dealing with liveness and Byzantine agreement[16, 5, 21].

3 Background

A distributed protocol $\mathcal{P}$ is an algorithm that makes computational steps and has an interface to receive and send external data, i.e. messages, input/output to the protocol, and time-related information. We analyse distributed protocols involving a set of agents $\mathcal{A}$ that interact by exchanging messages. In the following, we define the network that is used to exchange these messages, the types of faults that we expect from these agents, and some cryptographic assumptions.

Network model. We assume that agents have synchronised clocks, and they communicate through a point-to-point complete partially synchronous network. A partially synchronous network behaves asynchronously – that is, messages can be delayed arbitrarily – until an unknown point in time, called the Global Stabilisation Time, denoted by GST. From this point onwards, the network behaves synchronously, i.e., there is a known bound Δ on the message delay. Messages are always eventually delivered, and never duplicated.

Fault model. Non-faulty agents, namely, agents that follow the protocol, are called honest. Some other agents may present Byzantine faults and behave arbitrarily as a consequence. Our protocols tolerate an adversary that can corrupt agents statically, i.e., at the protocol onset.

Cryptographic assumptions. We rely on an asymmetric digital signature scheme to authenticate messages sent between agents. Moreover, we assume the existence of a public-key infrastructure (PKI) that identifies the cryptographic keys of agents in the protocol. Hence, we may use the identity of an agent in the place of a cryptographic key, and we implicitly assume in our algorithms that all messages are signed and checked for validity. We assume ideal cryptography, namely, that Byzantine agents cannot forge or tamper with digital signatures.

3.1 Agreement protocols

In this paper, we propose and analyse a number of *agreement protocols*. We use the term agreement protocol to broadly describe a distributed protocol where participants interact to select a common output value; this property is also sometimes called *consistency* in the literature. In the following, we precisely define a number of traditional agreement protocols, together with the properties that they must follow. They are later used in our exposition of hierarchical consensus protocols. For all our protocols, we require that both $n = |\mathcal{A}| \geq 2$ and $|\mathcal{V}| \geq 2$, where $|S|$ gives the cardinality of set S, we use $\mathcal{V}$ as the set of decision values of the protocol.

Reliable broadcast protocol (RBC). A reliable broadcast protocol is an agreement protocol where a special participant, called the *leader*, disseminates a value amongst the participating agents. Formally, a protocol $\mathcal{P}$ with participating agents $\mathcal{A}$, a known set of *valid* decision values $\mathcal{V}$, a leader agent $a_L \in \mathcal{A}$, and a leader input value $v_L \in \mathcal{V}$ is *a reliable broadcast protocol* if and only if the following properties hold:

- Safety:
  - Consistency: If two honest agents a and a' output values v and v', respectively, then $v = v'$.
  - Integrity: If the leader is honest, an honest agent cannot output v such that $v \neq v_L$.

- Liveness:
  - Validity: If the leader is honest, an honest agent eventually outputs a value.
  - Totality: If an honest agent outputs a value, all honest agents eventually output a value.

Note that this protocol is consistent, as any agreement protocol must be.

Consensus protocol. We will use consensus as a sub-protocol in our constructions. A consensus protocol is an agreement protocol where agents must eventually output a single agreed-upon value. Formally, a protocol $\mathcal{P}$ with participating agents $\mathcal{A}$, a chosen set of *valid* decision values $\mathcal{V}$, and where each agent $a_i \in \mathcal{A}$ has an input value $v_i \in \mathcal{V}$ is a *consensus protocol* if and only if the following properties hold:

- Safety:
  - Consistency: If two honest agents a and a' output values v and v', respectively, then $v = v'$.
  - Weak Validity: If all agents are honest and have the same input value v, then they can only output v.
- Liveness:
  - Termination: All honest agents eventually output a value.

In this work, we demonstrate how to use a hierarchical architecture to efficiently achieve consensus. Our techniques should be easily generalisable to a wide range of agreement protocols, but for simplicity and presentation purposes, we chose to focus on this weak version of consensus, as it is a versatile building block for other distributed applications and particularly state-machine replication.

Multi-threshold protocols. A protocol $\mathcal{P}$ is said to be t-resilient if it can tolerate up to t Byzantine faults, that is, in the presence of up to t Byzantine agents, the protocol can still satisfy its prescribed properties; t can also be seen as a *fault threshold* for the protocol. In this paper, we make use of the notion of multi-threshold protocols to take advantage of our hierarchical consensus architecture. Multi-threshold protocols have been first investigated by Blum *et al.* [2, 3, 4] and more comprehensively studied by Momose and Ren [22]. A *multi-threshold protocol* $\mathcal{P}$ has two fault thresholds t_{safe} and t_{live} such that, given a categorisation of its properties into Safety and Liveness, it must be the case that:

- $\mathcal{P}$ satisfies *at least* its Safety properties in the presence of at most t_{safe} faults;
- $\mathcal{P}$ satisfies *both* its Safety and Liveness properties in the presence of at most t_{live} faults.

Note that this definition implies that $t_{safe} \geq t_{live}$.

Momose and Ren [22] consider the resilience of the same protocol when executed in a synchronous network (with thresholds t^s_{safe} and t^s_{live}) or a partially synchronous network (with thresholds t^a_{safe} and t^a_{live}). They try to establish relations involving these four thresholds. In this paper, however, we only analyse (and use thresholds related to) partially synchronous protocols.

The usual theorems for Byzantine fault tolerance can be extended to multi-threshold protocols, with the main result being that *partially synchronous* multi-threshold agreement protocols require $t_{safe} + 2t_{live} < n$ [22], even when assuming the existence of digital signatures and a PKI.

Publicly verifiable protocols. The outputs of *publicly verifiable* protocols are endowed with *verifiability*, namely, outputs are intrinsically supported by a publicly verifiable proof. Given a predefined and protocol-specific publicly verifiable predicate $\mathit{Verify}(\pi, v)$ that checks whether π is a valid *proof* for output value v, output verifiability means that an honest agent

outputs v if and only if it has a proof π such that $\mathit{Verify}(\pi, v)$ holds. A consequence of this definition is that the reception of a proof by an honest agent is equivalent to outputting the associated supported value. So, verifiability can be used to achieve a sort of totality by relying on the propagation of a proof.

For instance, a *publicly verifiable reliable broadcast protocol* is defined as being an RBC protocol which has a publicly verifiable function $\mathit{Verify}(\pi, v)$, and, in addition to its Safety and Liveness properties, the following *Verifiability* property:

- Verifiability: An honest agent $a \in \mathcal{A}$ outputs $v \in \mathcal{V}$ if and only if it has a proof π such that $\mathit{Verify}(\pi, v)$.

The equivalence of outputting and obtaining a proof for a value arising from output *verifiability* in combination with the consistency property prevents honest agents from obtaining proofs supporting two different values for the same execution. These proofs allow third parties who do not take part in the protocol to validate its output: inconsistent proofs cannot be produced even by Byzantine agents. Verifiability is more of a *structural property* of the protocol as opposed to a safety property, but we place it in the Safety category for convenience as we expect it to be required for both failure thresholds t_{safe} and t_{live}. A publicly verifiable reliable broadcast protocol must respect the following inequality $t_{safe} + t_{live} < n$ [22], which also implies $t_{live} < \frac{n}{2}$ in our case.

4 Hierarchical Consensus: a framework for scalable BFT agreement

In this section, we present the notion of *hierarchical consensus* (HC). Intuitively speaking, this consensus protocol combines two agreement protocols: one optimistic that does not necessarily reach a decision, and a fallback one that is triggered if the optimistic fails to decide. The optimistic protocol is designed to come to a decision more efficiently than the fallback one. By giving up liveness in a strict sense (i.e., not requiring a decision to be eventually made), we can increase the resilience of our optimistic protocol – i.e., its ability to tolerate faulty agents – and, as a consequence, we can have a protocol that operates with a smaller number of agents if compared to a live consensus protocol as it is the case with the fallback.

4.1 Hierarchical consensus

We start off with a general definition of protocols that follows an hierarchical architecture, and we will refine this definition as needed for our results next. This definition uses Weak Validity, but it can be restated to use other validity notions, if preferred.

▶ **Definition 1.** *A protocol $\mathcal{P}$ with participating agents $\mathcal{A}$, a set $\mathcal{A}_l \subseteq \mathcal{A}$ of* live *agents, a known set of* valid *decision values $\mathcal{V}$, where each agent $a_i \in \mathcal{A}$ has an input value $v_i \in \mathcal{V}$, and a publicly verifiable function $\mathit{Verify}(\pi, v)$ is a* hierarchical consensus protocol *if and only if the following properties hold:*

- *Safety:*
  - *Consistency: If two honest agents $a, a' \in \mathcal{A}$ output values v and v', respectively, then $v = v'$.*
  - *Weak Validity: If all agents in $\mathcal{A}$ are honest and have the same input value v, an honest agent cannot output v' such that $v' \neq v$.*
  - *Verifiability: An honest agent $a \in \mathcal{A}$ outputs v if and only if it has a proof π such that $\mathit{Verify}(\pi, v)$.*
- *Liveness:*
  - *Termination: All honest agents in $\mathcal{A}_l$ eventually output a value.*

The hierarchical nature of this protocol appears in the separation of its agents. One can understand this definition as abstractly relying on two sub-protocols: one implemented by agents $\mathcal{A}_l$ and another implemented by agents in $\mathcal{A}_o = \mathcal{A} \setminus \mathcal{A}_l$, where $X \setminus Y$ denotes set difference between sets X and Y. The agents in $\mathcal{A}_o$ implement a not-necessarily-live (optimistic) sub-protocol, whereas agents in $\mathcal{A}_l$ implement a live (fallback) one, and the overarching definition enforces their safety, both locally (within these sub-protocols) and globally (across these sub-protocols).

If we consider the publicly verifiable version of HC and consensus, we can see that they can both be used to solve the other: agents in $\mathcal{A}_o$ can be guaranteed to terminate by simply waiting for $\mathcal{A}_l$'s output. This (feasibility) equivalence between HC and consensus implies, for instance, that bounds on failure thresholds for one definition must apply to the other. However, this equivalence does not imply that these two definitions must lead to protocols with the same executions, especially if we are interested in optimistic (i.e. best-case) executions. In this paper, we are interested in *optimistic* cases where HC can reach an agreement with less overhead compared to consensus. We are interested in the case where agents $\mathcal{A}_o$ finish of their own accord, as opposed to waiting for $\mathcal{A}_l$'s decision.

4.2 Handover

In this section, we devise a variant of hierarchical consensus where $\mathcal{A}_o$ is expected to output first.

▶ **Definition 2.** *A* handover protocol $\mathcal{P}$ *is a hierarchical consensus protocol with agents* $\mathcal{A}$*, live agents* $\mathcal{A}_l$*, and* $\mathcal{A}_o = \mathcal{A} \setminus \mathcal{A}_l$*, where communication between* $\mathcal{A}_o$ *and* $\mathcal{A}_l$ *is unidirectional from* $\mathcal{A}_o$ *to* $\mathcal{A}_l$*, namely, agents in* $\mathcal{A}_o$ *cannot receive messages from* $\mathcal{A}_l$*.*

We call it a handover protocol because, with the abstract view that $\mathcal{A}_o$ and $\mathcal{A}_l$ implement two sub-protocols, this protocol has to account for the safe handing over of control from $\mathcal{A}_o$ to $\mathcal{A}_l$. This definition allows agents in $\mathcal{A}_o$ to agree on a value and pass this decision on to agents in $\mathcal{A}_l$, at which point they can just propagate the agreed-upon value. However, agents in $\mathcal{A}_l$ must also be able to detect when agents in $\mathcal{A}_o$ are unable to come to a decision, at which point agents in $\mathcal{A}_l$ must take over and decide on a value. The intuitive idea behind this notion is that by lifting the (strict) *liveness* requirement for agents in $\mathcal{A}_o$, they can implement a (sub)protocol that is more efficient than a live one, while relying on a fallback (sub)protocol implemented by agents in $\mathcal{A}_l$ to rescue them when they are unable to come to a decision. One intuitive way to think about the hierarchical consensus and its handover counterpart is: the former is a flexible composition of two sub-protocols, whereas the latter is a type of chaining operator where the outputs of $\mathcal{A}_o$ are used by the agents in $\mathcal{A}_l$ to set their inputs.

4.3 Justifiable protocol

What types of protocols can agents in $\mathcal{A}_o$ implement in order to create a handover protocol? A safe *Justifiable Reliable Broadcast* (JRBC) protocol meets the requirements that we are after for the behaviour of the agents in $\mathcal{A}_o$. The (agents in this) protocol can output a decision, an indecision (i.e. the inability to reach a decision), and even a pre-decision; a guarantee that if the protocol decides, it must be on that pre-decided value. For a set of decision values $\mathcal{V}$, the outputs of agents are captured by the corresponding set of *output values* $\mathcal{O}$ that is a (disjoint) union of decision outputs on $\mathcal{V}$ (i.e. $\{dec.v \mid v \in \mathcal{V}\}$), the

pre-decision outputs on $\mathcal{V}$ (i.e. $\{pre.v \mid v \in \mathcal{V}\}$), and the indecision output $\{\bot\}$. We use a decision (and pre-decision) on v to denote $dec.v$ (and $pre.v$, respectively). We also require public verifiability. The addition of pre-decisions means that, in contrast to usual agreement protocols, Justifiable protocols can make *multiple* outputs. Of course, these outputs must still respect the properties of the protocol, e.g., consistency.

▶ **Definition 3.** *A protocol $\mathcal{P}$ with participating agents $\mathcal{A}$, a known set of* valid *decision values $\mathcal{V}$ and output values $\mathcal{O}$, a special designated* leader *agent $a_L \in \mathcal{A}$ with input value v_L, a publicly verifiable predicate Verify(π, o) for outputs, is a* Justifiable reliable broadcast *if and only if the following properties hold:*

- *Safety:*
  - *Consistency: If honest agents a and a' output decisions on v and v', respectively, then $v = v'$.*
  - *Pre-decision consistency: If honest agent a outputs a pre-decision on v and honest agent a' outputs a pre-decision or decision on v', then $v = v'$.*
  - *Indecision consistency: If honest agent a outputs $\bot$ and honest agent a' outputs o, then either $o = \bot$ or o is a pre-decision.*
  - *Verifiability: An honest agent a makes an output o if and only if it has a proof π such that Verify(π, o). o can be a decision, a pre-decision, or an indecision.*
  - *Integrity: If the leader is honest, an honest agent cannot output a decision on v such that $v \neq v_L$.*
  - *Pre-decision Integrity: If the leader is honest, an honest agent cannot output a pre-decision on v such that $v \neq v_L$.*
  - *Weak Termination: All honest agents eventually output (at least) a value. It can be a pre-decision, a decision, or an indecision.*
- *Liveness:*
  - *Validity: If the leader is honest, an honest agent eventually outputs a decision.*
  - *Totality: If an honest agent outputs a decision, all honest agents eventually output a decision.*

Note that *Indecision consistency and verifiability* implies that an indecision output by any agent is a proof of indecision for the protocol. That is, if an agent outputs an indecision value, it must be that no agent has outputted or will output a (verifiable) decision. In this definition, we have added *weak termination* as part of the Safety category of properties, even though this is, strictly speaking, a liveness property. This is to emphasise that *weak termination* is required to hold up to t_{safe} faults.

The ability to output a pre-decision releases agents from the obligation of reaching a strictly coordinated decision. The addition of pre-decision outputs allows the (safe version of the) protocol to separate executions reaching a potential decision from those that do not, to enable a safe handover, while being compatible with a weaker notion of liveness.

Impossibility theorems. We present two impossibility results that together show that Justifiability is a fundamental concept for handover protocols. First, we present a claim (that is more formally stated as Theorem 9 in Appendix A) that states that for any protocol P that can be used as a primary in a handover, P must satisfy some form of Justifiability.

▷ Claim 4. A handover protocol can only be constructed if the agents in $\mathcal{A}_o$ implement a justifiable protocol.

Furthermore, Justifiability adds a weak liveness requirement *even when* there are up to t_{safe} faults if compared to RBC protocols. Thus, an argument similar to the one demonstrating that a RBC protocol must satisfy $t_{safe} + 2t_{live} < n$ [22] can be applied to show that a JRBC protocol must satisfy $2t_{safe} + t_{live} < n$. As $t_{safe} \geq t_{live}$, the latter relation is strictly more restrictive than the former.

▶ **Theorem 5.** *A partially synchronous multi-threshold JRBC protocol must satisfy* $2t_{safe} + t_{live} < n$.

The proof is essentially the same as in [22], except that one of the executions that is required to be live with t_{live} faults is replaced with an execution that is weakly live with t_{safe} faults.

Proof. We proceed by contradiction. Let us assume that there is a partially synchronous multi-threshold Justifiable RBC protocol $\mathcal{P}$ such that $2t_{safe} + t_{live} = n$; the following proof can be trivially extended to the case $2t_{safe} + t_{live} > n$.

Let S, L_0 and L_1 be sets partitioning the set of agents $\mathcal{A}$, such that $|S| = |L_0| = t_{safe}$, $|L_1| = t_{live}$ and the sender a_L is in S. Let E, E_0 and E_1 be three executions as follows.

In E_0, agents in L_0 crashed and do not send any message, the remaining agents behave honestly, and a_L has input v_0. By weak liveness and t_{safe} faults, honest agents in E_0, including agents in L_1, eventually output either an indecision, a pre-decision on v_0, or a decision on v_0; let T_0 be the time when the last honest agent outputs in E_0. E_1 is the same as E_0 but with L_1 crashed instead of L_0 and a_L input being v_1 such that $v_1 \neq v_0$. By liveness and t_{live} faults, honest agents in E_1, including agents in L_0, eventually output a decision for v_1; let T_1 be the time when the last honest agent outputs in E_1.

In E, agents in S are Byzantine while the remaining agents behave correctly. Messages between L_0 and L_1 are not delivered until $GST = max(T_0, T_1)$, and agents in S behave towards each L_i for $i \in \{0, 1\}$ as in E_{1-i}, with the same respective network delays. Thus, agents in L_i cannot distinguish between E_{1-i} and S until GST, and so they must output as per E_{1-i}. So, agents in L_0 must output a decision on v_1 as per E_1, whereas agents in L_1 must output either an indecision, a pre-decision on v_0, or a decision on v_0 as per E_0. If an agent in L_1 outputs an indecision, we have a violation of Indecision consistency. Similarly, the output of a pre-decision (decision) on v_0 by an agent in L_1, would violate Pre-decision (Decision, respectively) consistency. Hence, $\mathcal{P}$ cannot be a JRBC protocol. ◀

Our proposed implementation for a partially synchronous JRBC, presented later, shows that this bound is tight. Given this bound, we have that when t_{safe} goes from $\frac{n}{3}$ to $\frac{n}{2}$, our JRBC protocol is limited to t_{live} going from $\frac{n}{3}$ to 0 whereas non-justifiable ones may have t_{live} going from $\frac{n}{3}$ to $\frac{n}{2}$ (and t_{safe} go above $\frac{n}{2}$ as well).

The notion of justifiability can also be applied to consensus protocols. Similarly to JRBC, Justifiable consensus protocols would have the three types of outputs (decisions, pre-decision, and indecision), the associated consistency, validity, verifiability, and weak termination notions. Unlike JRBC, it does not have an agent with the role of a leader and the properties associated with this role (e.g., integrity, totality, and termination based on leader honesty).

A Justifiable reliable broadcast can be constructed from a Justifiable consensus protocol in the typical way: the leader value is used to set up the initial values of the agents in the Justifiable consensus, and their consensus outputs become outputs for the Justifiable broadcast protocol construction. So, any impossibility for Justifiable reliable protocols must apply to Justifiable consensus protocols too.

5 Instantiating a hierarchical consensus protocol

5.1 Handover construction

In the following, we demonstrate a construction that creates a handover protocol by combining a JRBC protocol $\mathcal{P}_o$ with a consensus one $\mathcal{P}_l$. Intuitively speaking, the agents implementing $\mathcal{P}_l$ wait for the agents $\mathcal{P}_o$ to come to a decision. If they are unable to do so, the agents in $\mathcal{P}_l$ take over and decide. The intricate element in implementing this behaviour is ensuring that the handing over from agents in $\mathcal{P}_o$ to agents in $\mathcal{P}_l$ is safe.

▶ **Definition 6.** *Given a multi-threshold JRBC protocol $\mathcal{P}_o$ and a single-threshold consensus protocol $\mathcal{P}_l$, one can construct the handover protocol $\mathcal{P}^H(\mathcal{P}_l, \mathcal{P}_o)$ with agents $\mathcal{A}$, live agents $\mathcal{A}_l$ and remaining agents $\mathcal{A}_o = \mathcal{A} \setminus \mathcal{A}_l$ as follows:*

- *Agent $A_o \in \mathcal{A}_o$ runs $\mathcal{P}_o$ to obtain output o_o and proof π_o and then:*
  - *if o_o is a decision, it outputs o_o and sends o_o and associated proof π_o to all agents in $\mathcal{A}_l$;*
  - *if o_o is a pre-decision or indecision, it sends o_o and associated proof π_o to all agents in $\mathcal{A}_l$.*
- *The decision set for $\mathcal{P}_l$ is given by $\mathcal{V}_l = \{v \in \mathcal{V}_o \mid \mathit{Verify}(\pi_o, \mathit{pre}.v)\} \cup \{v \in \mathcal{V}_o \mid \mathit{Verify}(\pi_o, \perp)\}$, where $\mathcal{V}_o$ is the decision set of $\mathcal{P}_o$.*
- *An agent in $a_l \in \mathcal{A}_l$ behaves as follows:*
  - *Upon receiving a verifiable decision, it outputs this value and propagates it to agents in $\mathcal{A}_l$;*
  - *Upon receiving a verifiable pre-decision on v, it starts to behave as per $\mathcal{P}_l$ with input value v.*
  - *Upon receiving an indecision, it starts to behave as per $\mathcal{P}_l$ with input value v_i.*
  - *Upon $\mathcal{P}_l$ outputting some value v, output v.*

Note that the decision set $\mathcal{V}_l$ for $\mathcal{P}_l$ depends on the verifiable outputs of $\mathcal{P}_o$, namely, $\mathcal{P}_l$ can only decide on values that have been supported by the outputs of $\mathcal{P}_o$ as per $\mathcal{V}_l$. For instance, a proof of indecision allows agents in $\mathcal{P}_l$ to decide on any decision value in $\mathcal{V}_o$, whereas a pre-decision proof for v supports only the decision value v. To enforce that a decision value v for $\mathcal{P}_l$ conforms to $\mathcal{V}_l$, we can require that it be accompanied by a supporting proof π_o from $\mathcal{P}_o$. We omit this step from our construction for conciseness. Also, note that if $\mathcal{P}_l$ is publicly verifiable, we can make $\mathcal{P}^H$ publicly verifiable as well by outputting the proof π generated by $\mathcal{P}_l$.

The protocol $\mathcal{P}^H$ allows the agents $\mathcal{A}_o$ to decide and simply propagate their decision to the agents in $\mathcal{A}_l$; this sort of *optimal* behaviour is the main motivation for proposing handover protocols. Of course, in the worst-case scenario, if the agents in $\mathcal{A}_o$ are unable to reach a decision, the consensus protocol $\mathcal{P}_l$ has to run in addition to $\mathcal{P}_o$. Note that the protocol $\mathcal{P}^H$ does not output the indecision or pre-decisions output by $\mathcal{P}_o$; it merely uses them internally as part of the handover mechanism we devise. Thus, only decisions are required to be publicly verifiable. Also, for convenience, we have that the execution of $\mathcal{P}_o$ by an agent returns an output and a proof.

We show that $\mathcal{P}^H$ is indeed a handover protocol next.

▶ **Theorem 7.** *For a* safe *Justifiable reliable broadcast protocol $\mathcal{P}_o$ with safety threshold t^w_{safe} and a consensus protocol $\mathcal{P}_l$ with liveness threshold t^l_{live}, the protocol $\mathcal{P}^H(\mathcal{P}_l, \mathcal{P}_o)$ is a handover protocol.*

Proof. The messages sent between $\mathcal{A}_o$ and $\mathcal{A}_l$ flow only from the former to the latter. So, we only need to demonstrate that $\mathcal{P}^H(\mathcal{P}_l, \mathcal{P}_o)$ satisfies all the properties of a hierarchical consensus protocol.

- Consistency: $\mathcal{P}_o$ and $\mathcal{P}_l$ enforce local consistency, namely, no two agents within these protocols can decide on different values. The definition of $\mathcal{V}_l$ predicated on the outputs of $\mathcal{P}_o$ ensures inter-protocol consistency. Let us assume that an agent $a_o \in \mathcal{A}_o$ outputs a decision on v'. By the consistency properties of $\mathcal{P}_o$, it must be that all agents in $\mathcal{A}_o$ output either a pre-decision or a decision on that value. Hence, the decision set $\mathcal{V}_l$ for agents in $\mathcal{A}_l$ can contain only the value v'. Therefore, $\mathcal{P}_l$ can only output v'.
- Weak Validity: If all the agents in $\mathcal{A}_o$ are honest and have the same input value v', if they output a pre-decision or decision, it must be on v', by their integrity properties. So, the agents in $\mathcal{A}_l$ receive from agents in $\mathcal{A}_l$ enough messages to support v' as their initial value or enough indecision so they use their own initial value, which is also v'. By $\mathcal{P}_l$ own notion of validity, we have that agents in $\mathcal{A}_l$ can only output v' too.
- Verifiability: Decisions of $\mathcal{P}^H$ are verifiable by relying on the publicly verifiable functions of the underlying protocols $\mathcal{P}_o$ and $\mathcal{P}_l$.
- Termination: If an agent in $\mathcal{A}_o$ outputs a decision, it will eventually be propagated to all agents in $\mathcal{A}$. Otherwise, by weak termination of $\mathcal{P}_o$, each honest agent in $\mathcal{A}_l$ is guaranteed to eventually receive either a pre-decision or an indecision (with accompanying proof) from agents in $\mathcal{A}_o$. Therefore, all honest agents in $\mathcal{A}_l$ will eventually run $\mathcal{P}_l$ with some input value. By termination of $\mathcal{P}_l$, they all eventually output the value from $\mathcal{P}_l$. ◀

Note that we do not use the (strong) liveness property of JRBC to show Termination for $\mathcal{P}^H$. This property is proven using only JRBC's weak liveness. On the other hand, JRBC's Liveness guarantees that *when* t_{live} is met, the optimistic protocol will succeed, provided that the timeout is large enough.

To complete the construction of a handover protocol, we propose a partially synchronous JRBC protocol implementation; many well-known consensus protocol implementations exist, of course.

5.2 Partially synchronous JRBC (psync-JRBC)

Algorithm 1 depicts the behaviour of an honest agent for a JRBC protocol, which we call *psync-JRBC*. This protocol behaves similarly to a traditional RBC protocol, but it relies on a timeout to ensure agents output a value even if they are yet unaware of a decision being reached. The timeout should be tuned to the network delay as best as possible, but note that its value only impacts performance.

Psync-JRBC is implemented in two rounds of voting to ratify the proposal of the leader; the message $(\textsc{Proposal}, v)$ denotes the proposal of value v. In the first round, agents use message $(\textsc{Prepare}, v)$ to ratify the value v. When an agent receives a quorum of containing more than half of distinct honest agents Prepare-ratifying the same value v, it has obtained a *pre-decision* for v; these Prepare messages form a proof for this pre-decision. Reaching this quorum (before a timeout) triggers the second round of voting, which relies on message $(\textsc{Commit}, v)$ to ratify v the pre-decided value v. A quorum including $2t_{safe}+1$ honest agents Commit-ratifying the same value v means that a decision has been reached; these Commit messages represent a proof for a decision. If the timeout happens before a pre-decision is obtained, the agent aborts by sending an Abort message. A quorum of $n - 2t_{safe}$ abort messages from distinct honest agents represents a proof of indecision. If the timeout happens

Algorithm 1 Implementation of protocol *psync JRBC*.

local variables:
$\mathcal{A}$. The protocol agents.
v_i. The agent input.
a_i. The agent identity.
a_L. The leader identity.
timer. An object that expires after some timeout duration.
procedure *start()*
  start *timer*;
  if $a_i = a_L$ **then**
    send $(\textsc{Proposal}, v_i)$ to $\mathcal{A}$;
end
upon *receiving* $(\textsc{Proposal}, v)$ *from* a_L **do**
  if $(\textsc{Prepare}, _)$ *not sent* **then**
    send $(\textsc{Prepare}, v)$ to $\mathcal{A}$;
end
upon *receiving* $(\textsc{Prepare}, v)$ *from* $T_p = \left\lceil \frac{t_{safe}+n+1}{2} \right\rceil$ *distinct senders* **do**
  if *timer not expired* **then**
    send $(\textsc{Commit}, v)$ to $\mathcal{A}$;
end
upon *receiving* $(\textsc{Commit}, v)$ *from* $T_d = 2t_{safe} + 1$ *distinct senders* **do**
  send $(dec.v, \pi_d)$ to $\mathcal{A}$;
  output $dec.v$;
end
upon *receiving* $(\textsc{Abort})$ *from* $T_a = n - t_{safe}$ *distinct senders* **do**
  send $(\perp, \pi_i)$ to $\mathcal{A}$;
  output $\perp$;
end
upon *timer expires* **do**
  if $(\textsc{Commit}, _)$ *not sent* **then**
    send $\textsc{Abort}$ to $\mathcal{A}$;
  if $(\textsc{Commit}, v)$ *was sent for some* v **then**
    send $(pre.v, \pi_p)$ to $\mathcal{A}$;
    output $pre.v$;
  end
end
upon *receiving* (o, π) *where* $Verify(\pi, o)$ **do**
  send (o, π) to $\mathcal{A}$;
  output o;
end

after a pre-decision, the agent outputs the pre-decision. The agent outputs a decision or an indecision as soon as it has obtained a proof for it. We assume that signature creation and verification are being carried out as messages are sent and received.

▶ **Theorem 8.** *The protocol psync-JRBC is a JRBC protocol with thresholds* $t_{safe} \in \left[0, \left\lfloor \frac{n-1}{2} \right\rfloor\right]$ *and* $t_{live} = \min(\{n - T_p, n - T_d, t_{safe}\})$*, where* min *returns the minimal element of the set.*

Proof. We show that psync-JRBC satisfies all Safety properties of a JRBC protocol for $t_{safe} \in \left[0, \left\lfloor \frac{n-1}{2} \right\rfloor\right]$ and both Safety and Liveness properties for $t_{live} = \min(\{n - T_p, n - T_d, t_{safe}\})$.

- Consistency: We prove this by contradiction. So, let us assume that two honest agents output decisions on v and v' such that $v \neq v'$. For each of these decisions, at least one honest agent must have $\textsc{Commit}$ for each of these values. Let us call them a and a' for values v and v', respectively. Thus, a and a' must have obtained pre-decisions for v and v', respectively. However, a pre-decision requires a quorum of $T_p = \left\lceil \frac{t_{safe}+n+1}{2} \right\rceil$ agents, namely, it requires the participation of more than half of the honest agents. Hence, by a quorum intersection argument, it must be that an honest agent has sent a $\textsc{Prepare}$ message for both values v and v', which contradicts our algorithm.
- Pre-decision consistency: We can use the pre-decision quorum intersection argument again to show here that two pre-decisions cannot be obtained for two distinct values. We can prove the other case, i.e., that a decision and a pre-decision output by honest agents cannot support different values, using a similar argument. Let us assume that agent a has obtained a pre-decision for v and that agent a' has obtained a decision for v'. The quorum required for a decision must include at least one honest agent; let us call this agent a^*. This agent must have, then, obtained a pre-decision for v'. That contradicts the fact that two pre-decisions for distinct values cannot be obtained for the same run of the protocol.
- Indecision consistency: We prove this by contradiction. Let us assume that agent a outputs $\perp$ and agent a' outputs a decision on v. By our algorithm, for a to output $\perp$, it must be that $n - 2t_{safe}$ distinct honest agents have aborted. For the decision on v, by

our algorithm, there must have been $t_{safe}+1$ distinct honest agents who have obtained a pre-decision for v. There are $T_d + T_a = (n - t_{safe} + 2t_{safe} + 1 - n) = t_{safe} + 1$ overlapping agents in the intersection of these two quorums, but an honest agent cannot both abort and possess a pre-decision, a contradiction.

- Decision, Pre-decision, and indecision verifiability: A proof of a decision is a collection of $T_d = 2t_{safe}+1$ COMMIT messages from distinct agents; a proof of a pre-decision is a set of $T_p = \left\lceil \frac{t_{safe}+n+1}{2} \right\rceil$ PREPARE messages from distinct agents; and a proof of indecision is a set of $T_a = n - t_{safe}$ ABORT messages from distinct agents. Given that $t_{live} \leq t_{safe}$, to satisfy the public verifiability requirement of $t_{safe} + t_{live} < n$, it must be that $t_{safe} \in \left[0, \left\lfloor \frac{n-1}{2} \right\rfloor\right]$.
- Integrity: If the leader a_L is honest, honest agents can only send PREPARE messages for v_L. Given that a pre-decision quorum requires the participation of honest agents, a pre-decision can only be for v_L and, as a consequence, a decision can only be for v_L.
- Pre-decision integrity: As per the previous property.
- Weak termination: The timer of honest agents will eventually expire. When that happens, two cases can arise. If an honest agent has obtained a pre-decision, it will propagate this pre-decision, eventually compelling all honest agents to output this pre-decision. If no honest agent has obtained a pre-decision, they will all send abort messages. Given that there are $n - t_{safe}$ honest agents, they can eventually obtain an abort proof and output $\perp$.
- Termination: Our requirement $t_{live} = \min(\{n - T_p, n - T_d, t_{safe}\})$ means that we have enough honest agents to meet the quorum for a pre-decision and a decision. Hence, if the leader is honest and the timer lasts for enough time, the algorithm will converge after GST to a decision on the leader's input value. ◀

5.3 Putting it all together

With our handover construction $\mathcal{P}^H$ on Def. 6 and our psync-JRBC protocol in Alg. 1 used as $\mathcal{P}_o$, we have a protocol that relies on the creation of committees for $\mathcal{A}_o$ and $\mathcal{A}_l$ with the appropriate failure thresholds. The benefit of our approach is apparent when these collections of agents are obtained via *random sampling with repetition*, i.e., when they are selected randomly and independently from an underlying (larger) population of agents. The precise mechanism by which agents are sampled is irrelevant to our construction, as long as each agent has a probability $p > \frac{2}{3}$ to be honest.

Stochastic reasoning. The higher fault tolerance for JRBC (i.e. primary) protocols allows them to rely on small committees if compared to typical consensus protocols. In this section, we dive into a probabilistic approach to choose committees to implement (i.e. run) our *deterministic* protocols. That does not make our protocols stochastic, but that means that our correctness guarantees become probabilistic. We rely on a notion of *stochastic reasoning* to bound the probability with which correctness may be violated. We use the error parameter ϵ as the upper bound on correctness violation, which should be negligible in the security parameter. In our context, a violation arises if we select a committee with the wrong proportion of honest agents. This error parameter can be understood as the probability with which we will select a committee with the wrong proportion of honest agents.

We select agents for our committees from a population of agents with a probability p of being honest. We rely on a process that selects randomly and independently (with repetition) n agents from this population. The probability that this set has k honest agents is governed by a binomial distribution with parameters (n, p). The probability that this collection has up to k

honest agents is equal to the cumulative distribution $F(k;n,p) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$. This probability is equivalent to the probability of the collection having at least $n-k$ Byzantine agents, and so the probability that it has at most $n-k-1$ Byzantine agents is $1-F(k;n,p)$. So, by minimising $F(k;n,p)$, we increase the probability that $n-k-1$ bounds the number of Byzantine agents in this collection, and $F(k;n,p)$ can be seen as the probability that $n-k-1$ is not a correct bound. We use $K(n,p,\epsilon) = max(\{k \in \{0 \dots n\} \mid F(k;n,p) \leq \epsilon\})$ to find the k, if it exists, that represents the tightest $k+1$ lower bound to the number of honest agents in the committee, respecting the error probability ϵ. For a given probability p of agents to be honest, a target maximum number of Byzantine fault t and error threshold ϵ, we compute the committee size $n^\epsilon(p,t)$ as the smallest number, if it exists, that guarantees less than t faults, i.e., $n^\epsilon(p,t) = \min(\{n \in \mathbb{N} \mid k = K(n,p,\epsilon) \wedge n-k-1 \leq t\})$.

In Table 1, we illustrate how the ability to tolerate more Byzantine agents can lead to smaller committee sizes. We present the values of $n^\epsilon(p,t)$ for some values of p and ϵ and for failure thresholds $t = \lfloor \frac{n-1}{2} \rfloor$ and $t = \lfloor \frac{n-1}{3} \rfloor$; we choose 0.68 as a starting value for p because of the need for a 2/3 ratio of honest agents to implement consensus. These values demonstrate the substantial impact that tolerating more failures can have in the size of the committees being randomly selected. For instance, for $p = 0.68$ and $\epsilon = 10^{-14}$, a primary protocol would require a committee of size 411, whereas the secondary protocol would require the participation of 72061 agents; the secondary committee size would be about 17500% that of the primary. As expected, the discrepancy between committee sizes reduces as the probability of an agent being honest increases.

Communication cost. We examine the communication complexity of handover protocols, i.e. the number of bits sent by honest agents. We note $\mathcal{C}^P(n)$ the communication cost of protocol P as a function of the number of participating agents. It is straightforward to see that $\mathcal{C}^{\mathcal{P}^H}(|\mathcal{A}|) = \mathcal{C}^{\mathcal{P}_o}(|\mathcal{A}_o|) + \mathcal{C}^{\mathcal{P}_l}(|\mathcal{A}_l|) + O(|\mathcal{A}_o||\mathcal{A}_l|)$ in general, and for optimistic executions $\mathcal{C}^{\mathcal{P}^H}(|\mathcal{A}|) = \mathcal{C}^{\mathcal{P}_o}(|\mathcal{A}_o|) + O(|\mathcal{A}_o||\mathcal{A}_l|)$. This shows that there are gains to be expected from the optimistic executions, although the handover can, in some situations, incur a cost higher than just running the fallback. More generally, since our technique yields a constant factor improvement in the committee size, we cannot analyse them using asymptotic notation. Depending on the protocols at hand, tail bounds on the binomial distribution (e.g., the Chernoff bound) may be used instead. Increasing t_{safe} for $\mathcal{P}_o$ makes $|\mathcal{A}_o|$ smaller, at the expense of decreasing t_{live} and therefore the probability of making $\mathcal{P}_o$ (strongly) live.

We intended to use our hierarchical protocol as a consensus engine for blockchains. In this context, we could set up incentive structures that can be naturally implemented in these distributed systems to encourage optimistic executions. For instance, they could financially reward more favourably these executions against fallback ones. These incentive structures

Table 1 Tables depicting the committee size $n^\epsilon(p,t)$ for different values of p and ϵ contrasting the size requirements for committees tolerating $\lfloor \frac{n-1}{2} \rfloor$ and those tolerating $\lfloor \frac{n-1}{3} \rfloor$.

$\epsilon = 10^{-10}$		
p	$\lfloor \frac{n-1}{2} \rfloor$	$\lfloor \frac{n-1}{3} \rfloor$
0.68	281	49795
0.72	179	2944
0.76	123	901
0.8	87	403
0.84	61	214
0.88	45	124
0.92	31	70

$\epsilon = 10^{-14}$		
p	$\lfloor \frac{n-1}{2} \rfloor$	$\lfloor \frac{n-1}{3} \rfloor$
0.68	411	72061
0.72	265	4276
0.76	179	1303
0.8	125	592
0.84	89	313
0.88	65	181
0.92	45	106

$\epsilon = 10^{-18}$		
p	$\lfloor \frac{n-1}{2} \rfloor$	$\lfloor \frac{n-1}{3} \rfloor$
0.68	543	94366
0.72	351	5608
0.76	237	1720
0.8	167	769
0.84	121	415
0.88	87	238
0.92	59	142

can be used to optimise another aspect of our construction. Note that when $t_{safe} > t_{live}$ for $\mathcal{P}_o$, it only takes a small number of Byzantine agents ($n - 2t_{safe}$) to prevent the primary from reaching a decision. For instance, for $n = 2t_{safe} + 1$, it only takes one Byzantine agent (being silent, for example) to prevent $\mathcal{P}_o$ from being live. Of course, weak liveness still holds for the primary, so the Byzantine agents in $\mathcal{P}_o$ can delay a $\mathcal{P}^H$ decision, but they cannot prevent it. Thus, once more, incentive structures can be set up to ensure that delaying a decision is not worthwhile, encouraging, then, the Byzantine agents in $\mathcal{A}_o$ to cooperate in reaching a $\mathcal{P}_o$ decision.

6 Conclusion

In this paper, we introduce the notion of *hierarchical consensus* as a framework to create efficient BFT consensus protocols by combining an optimistic efficient primary sub-protocol that is safe and only weakly live with a fallback (not as efficient) secondary protocol that is safe and live. Giving away (strict) liveness is the price paid for the efficiency of the primary protocol.

We propose the notion of a *Justifiable RBC* protocol, and *justifiability* more generally, as a way to characterise primary sub-protocols: it captures the safety and weak liveness requirements necessary for a safe handover to the secondary sub-protocol. We show that for a Justifiable RBC protocol, it must be that $2t_{safe} + t_{live} < n$; this inequality should be a useful guide for designers of such protocols. Moreover, we devise a JRBC implementation that shows that this is a tight bound.

We also introduce the *handover construction* $\mathcal{P}^H$ that creates a hierarchical consensus protocol by combining a Justifiable RBC protocol $\mathcal{P}_o$ and a consensus protocol $\mathcal{P}_l$. This construction can combine our *psync JRBC* implementation with a consensus implementation of choice to create a hierarchical consensus implementation. We show that JRBC protocols can tolerate up to $\lfloor \frac{n-1}{2} \rfloor$ faults while remaining safe and weakly live as opposed to the $\lfloor \frac{n-1}{3} \rfloor$ failure tolerance level for live consensus protocols; this difference is crucial to achieve efficiency.

We employ a notion of *stochastic reasoning* to demonstrate the impact that these levels of tolerance have on the size of committees implementing $\mathcal{P}_o$ and $\mathcal{P}_l$. Our calculation show that, for a very small ϵ and p close to 0.68 (i.e. the probability of drawing an honest agent is close for 2/3), the size for a committee where $t = \lfloor \frac{n-1}{2} \rfloor$ is in the hundreds of agents, whereas for $t = \lfloor \frac{n-1}{3} \rfloor$ it is usually in the thousands. This demonstrates the significant efficiency gains that can be achieved by our framework.

Future work. The area of hierarchical consensus seems very rich and not yet well explored. We intend to analyse the handover mechanisms in [25, 26, 24] within our framework. Additionally, while we provide a single instantiation of our framework, we believe there are many more that could be of interest. One could consider a handover to another optimistic protocol instead of a secondary. This could be retried some number of times or indefinitely, depending on the efficiency of the various protocols involved. Similarly, so far we have only instantiated a Justifiable Reliable Broadcast, whereas Justifiability could be added to consensus and its variants, e.g., to avoid the reliance on leaders. Going further, we plan to enhance the modularity of our framework by making the notion of Justifiability generically applicable to any agreement protocol, as well as to remove any specific notion of Validity for consensus and hierarchical consensus, instead relying on the insights from Civit et al. [9]. Adding to the theoretical exploration of Justifiability, we expect to complete its characterisation with

tighter feasibility results, and to show to possibility of compiling any agreement protocol into a Justifiable one.

Other avenues of research include taking full advantage of multi-threshold protocols for more resilient primary and/or secondary, as well as better analysis to determine the best trade-off between t_{safe} and the probability to make the protocol live, depending on the protocols at hand. In the context of using our hierarchical protocol as a consensus engine for blockchains, we plan to more formally analyse the incentive structures that can be put in place to encourage optimistic executions. Our architecture is particularly suited to this because we can readily detect optimistic executions simply by observing the provenance of the output.

References

1 Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 381–391. ACM, 2022. doi:10.1145/3519270.3538426.

2 Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 131–150. Springer, 2019. doi:10.1007/978-3-030-36030-6_6.

3 Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 547–572. Springer, 2021. doi:10.1007/978-3-030-92075-3_19.

4 Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 707–731. Springer, 2020. doi:10.1007/978-3-030-56880-1_25.

5 Manuel Bravo, Gregory V. Chockler, and Alexey Gotsman. Making byzantine consensus live. *Distributed Comput.*, 35(6):503–532, 2022. doi:10.1007/S00446-022-00432-Y.

6 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002. doi:10.1145/571637.571640.

7 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 145–156. ACM, 2024. doi:10.1145/3662158.3662792.

8 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. All byzantine agreement problems are expensive. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 157–169. ACM, 2024. doi:10.1145/3662158.3662780.

9 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. On the validity of consensus. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed*

Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023, pages 332–343. ACM, 2023. doi:10.1145/3583668.3594567.

10 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Make every word count: Adaptive byzantine agreement with fewer words. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, volume 253 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.OPODIS.2022.18.

11 Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 683–696. ACM, 2022. doi:10.1145/3548606.3559375.

12 Alfonso de la Rocha, Lefteris Kokoris-Kogias, Jorge M. Soares, and Marko Vukolic. Hierarchical consensus: A horizontal scaling framework for blockchains. In *42nd IEEE International Conference on Distributed Computing Systems, ICDCS Workshops, Bologna, Italy, July 10, 2022*, pages 45–52. IEEE, 2022. doi:10.1109/ICDCSW56584.2022.00018.

13 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 623–653. Springer, 2021. doi:10.1007/978-3-030-90459-3_21.

14 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

15 Fatima Elsheimy, Giorgos Tsimos, and Charalampos Papamanthou. Deterministic byzantine agreement with adaptive $O(n \cdot f)$ communication. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1120–1146. SIAM, 2024. doi:10.1137/1.9781611977912.43.

16 Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. *ArXiv*, abs/2106.10362, 2021. arXiv:2106.10362.

17 Yuval Gelles and Ilan Komargodski. Scalable agreement protocols with optimal optimistic efficiency. In *Security and Cryptography for Networks - 14th International Conference, SCN 2024, Amalfi, Italy, September 11-13, 2024, Proceedings, Part I*, volume 14973 of *Lecture Notes in Computer Science*, pages 297–319. Springer, 2024. doi:10.1007/978-3-031-71070-4_14.

18 Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement. *Math. Syst. Theory*, 26(1):41–102, 1993. doi:10.1007/BF01187074.

19 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. URL: https://eprint.iacr.org/2016/889.

20 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

21 Andrew Lewis-Pye, Joachim Neu, Tim Roughgarden, and Luca Zanolini. Accountable liveness. *arXiv preprint arXiv:2504.12218*, 2025.

22 Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In *CCS '21, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1686–1699. ACM, 2021. doi:10.1145/3460120.3484554.

23 S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

24 A. W. Roscoe. *Understanding Decentralised Systems: Fundamentals and Blockchain*. Springer, 2026 (to appear).

25 A. W. Roscoe, Pedro Antonino, and Jonathan Lawrence. *The Consensus Machine: Formalising Consensus in the Presence of Malign Agents*, pages 136–162. Springer Nature Switzerland, Cham, 2023. doi:10.1007/978-3-031-40436-8_6.
26 A. W. Roscoe, Pedro Antonino, and Jonathan Lawrence. *Abstracting and Verifying Decentralised Systems in CSP*, pages 172–202. Springer Nature Switzerland, Cham, 2024. doi:10.1007/978-3-031-67114-2_8.
27 G. Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/Paper.pdf.

A Impossibility Theorem for Justifiability

Before stating our proof, we add some slight precisions to our execution model. An execution E of the protocol P is represented by a potentially infinite *list* of *events*. Events in E are accessed by an index $i \in \mathbb{N}$, that we also call *point* or *instant*. An event may be one of the following: sending/reception of a message by some agent (Byzantine or not), local input/output to P by some honest agent, and the emission of a tick event measuring the passage of time. We say that E can be *completed* into another execution E' at a point $i \in \mathbb{N}$ if E and E' are both executions of P and for $k \leq i$, their kth events are equal. We call an execution *terminating* if some honest agent makes an output. We say that a pre-decision exists in an execution E, if there is a point i in that execution where all completions of E from i that terminate decide the same value v_E.

▶ **Theorem 9.** *Let H^P be a protocol that takes as a parameter an input protocol P, and the nodes in $\mathcal{A}_w$ must only execute P. P is also allowed to send external messages, e.g., to agents in A_l, and those messages are implicitly sent to A_w as well. If H^P is a handover protocol, then we can build a protocol P' that is exactly P with the addition of local output, such that P' must eventually output a proof that there exists a pre-decision value.*

Proof. Without loss of generality, we will consider H^P to be a protocol with a single agent in $\mathcal{A}_l$ that is always honest. This is possible because, if there cannot be a handover from P using a trusted party, then no handover protocol such as H^P can exist. We name the agent $a_l \in \mathcal{A}_l$.

Assume that there is an execution X of P, where:

1. No honest agent ever makes an output in X.
2. At any point i in X, X can be completed into two other executions X_0^i and X_1^i, such that in X_0^i and X_1^i, all honest agents outputs v_0 (respectively, v_1), and $v_0 \neq v_1$.

Then, in an execution of H^P where P is running X, H^P must eventually output a value v_h at some point i in X. At that point, X can be completed into X_j^i or X_j^i, with $j \in \{0, 1\}$ such that $v_h \neq v_j$. This is a violation of Consistency for H^P, therefore such X does not exist.

The non-existence of X can be precisely stated as follows: For all executions E of P, either

1. Some honest agent a eventually makes an output in E.
2. There is a point i in E, such that for all pairs of completions of E E_0^i and E_1^i, either one of those do not make an output, or they output the same value v.

This can be simplified to say that the following statement holds: for all of P's executions E, if E does not terminate, then there is a point i in E such that all completions of E from i that do terminate must all output the same value noted v_E, i.e. all non terminating executions have a pre-decision. Because all terminating executions have a pre-decision, we conclude that all executions have a pre-decision.

We have shown that P must reach a state where a pre-decision exists, but we have not shown that honest nodes must know when that has happened. We do so below.

Consider an execution E of P, and let i be the earliest instant (i.e. the smallest index) in E when there is a pre-decision noted v_E. Because of Liveness, a_l have to eventually output a value v, let j be the instant when that happens. If $j < i$, then there is either no pre-decision at that instant, or there is a completion from j with a different decision. Both statements imply that there is an additional completion of E from j that output a different values $v' \neq v_E$. Either $v' \neq v$ or $v_E \neq v$, hence one of those executions creates a contradiction with consistency. Therefore, $j \geq i$.

In other words, for all executions E, a_l will eventually know that a pre-decision exists. Building P' is therefore trivial: Run a_l's code along with P. Whenever a_l outputs a value, we are guaranteed that a pre-decision exists. The messages fed to a_l's code constitute the proof of this fact, ◀

On the Shape Containment Problem Within the Amoebot Model with Reconfigurable Circuits

Matthias Artmann
Paderborn University, Germany

Andreas Padalkin
Paderborn University, Germany

Christian Scheideler
Paderborn University, Germany

Abstract

In *programmable matter*, we consider a large number of tiny, primitive computational entities called *particles* that run distributed algorithms to control global properties of the particle structure. *Shape formation* problems, where the particles have to reorganize themselves into a desired shape using basic movement abilities, are particularly interesting. In the related *shape containment* problem, the particles are given the description of a shape S and have to find maximally scaled representations of S within the initial configuration, without movements. For example, if S is a triangle, they have to identify the largest subsets of particles that already form a triangle. While the shape formation problem is being studied extensively, no attention has been given to the shape containment problem, which may have additional uses besides shape formation, such as detecting structural flaws.

In this paper, we consider the shape containment problem within the *geometric amoebot model* for programmable matter, using its *reconfigurable circuit extension* to enable the instantaneous transmission of primitive signals on connected subsets of particles. We first prove a lower runtime bound of $\Omega(\sqrt{n})$ synchronous rounds for the general problem, where n is the number of particles. Then, we present simple and efficient primitives for identifying subsets that form the desired shape. Using these primitives, we construct a large class of shapes which we call *snowflakes*. This class contains, among others, all shapes composed of parallelograms and hexagons, and the class of *star convex* shapes. Let k be the maximum scale of the considered shape in a given amoebot structure. If the shape is star convex, we solve it within $\mathcal{O}(\log^2 k)$ rounds. If it is a snowflake but not star convex, we solve it within $\mathcal{O}(\sqrt{n}\log n)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Self-organization; Theory of computation → Computational geometry

Keywords and phrases Programmable matter, amoebot model, reconfigurable circuits, shape containment

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.7

Related Version *Full Version*: `https://arxiv.org/abs/2501.16892` [4]

Funding This work was supported by the DFG Project SCHE 1592/10-1.

Acknowledgements We thank Daniel Warner for his guidance and helpful discussions.

1 Introduction

Programmable matter envisions a material that can change its physical properties in a programmable fashion [25] and react to external stimuli. It is typically viewed as a system of many identical micro-scale computational entities called *particles*. Potential application areas include minimally invasive surgery, maintenance, exploration, and manufacturing. While significant progress is being made in the field of micro-scale robotics [5, 26], the fundamental capabilities and limitations of such systems are studied in theory using various models [24].



© Matthias Artmann, Andreas Padalkin, and Christian Scheideler;
licensed under Creative Commons License CC-BY 4.0



In the *amoebot model* of programmable matter, the particles are called *amoebots* and are placed on the nodes of a graph. We assume that the occupied nodes form a connected subgraph. Since information can only travel through the edges of the graph, there is a natural lower bound of $\Omega(D)$ for many problems, where D is the diameter of this subgraph.

Motivated by this, we consider the *reconfigurable circuit extension* of the model, which allows better results with reasonable modifications. In this extension, the amoebots can construct simple communication networks called *circuits* on connected subgraphs of the structure and broadcast primitive signals on these circuits instantaneously. This allows polylogarithmic solutions to many problems, e.g., leader election, consensus, shape recognition, and shortest path forest construction [12, 20, 21].

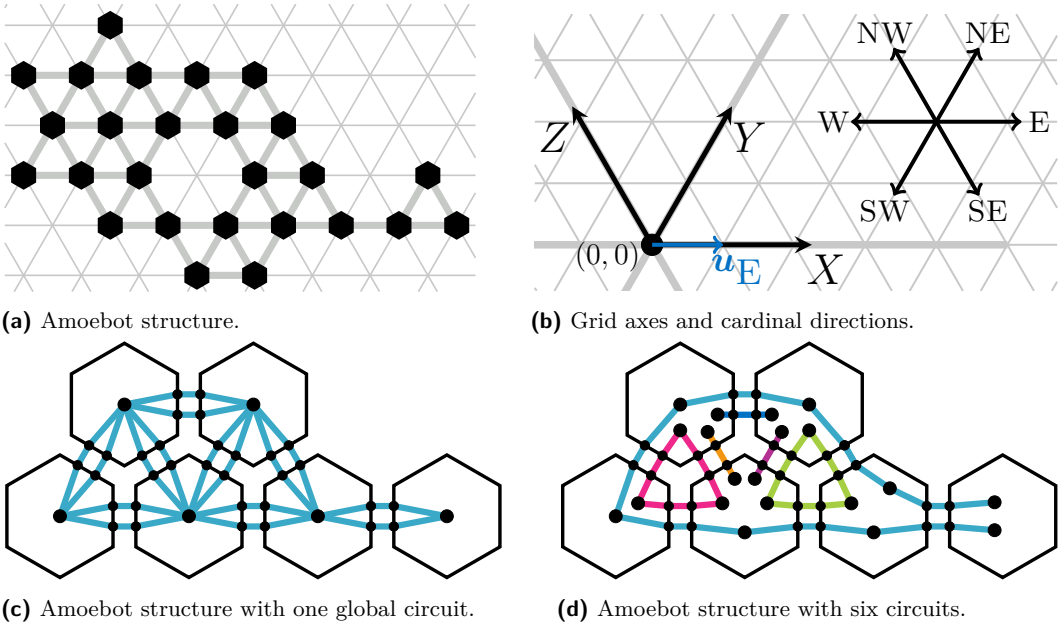
The *shape formation problem*, where the amoebot structure has to reconfigure itself into a given shape, is a standard problem of particular interest [24]. In this paper, we study the related *shape containment problem*: Given a shape S, the amoebots must find the maximum scale at which S can be placed within their structure and identify all valid placements at this scale. This can be useful for shape formation by *self-disassembly*, i.e., disconnecting all amoebots that are not part of a selected placement of the shape from the structure [13, 14]. The problem can also be interpreted as a discrete variant of the *polygon containment problem* in classical computational geometry, which has been studied extensively [6, 23]. To our knowledge, there are no distributed solutions that apply to the amoebot model.

1.1 Geometric Amoebot Model

We use the *geometric amoebot model* for programmable matter, as proposed in [9]. Using the terminology from the recent *canonical* model description [8], we assume *common direction* and *chirality*, *constant-size memory*, and a fully synchronous scheduler, making it *strongly fair*. We describe the model in sufficient detail here and refer to [8, 9] for more information.

The geometric amoebot model places n particles called *amoebots* on the infinite regular triangular grid graph $G_\Delta = (V_\Delta, E_\Delta)$ (see Fig. 1a). This is the typical graph used in the amoebot literature [9, 12, 19–21]. Each amoebot occupies one node, and each node is occupied by at most one amoebot. We identify each amoebot with the grid node it occupies to simplify the notation. Thereby, we define the *amoebot structure* $A \subset V_\Delta$ as the set of occupied nodes. We assume that A is finite and its induced subgraph of G_Δ is connected.

Each amoebot has a local *compass* identifying one of its incident grid edges as the East direction, and a *chirality* defining its local sense of clockwise rotation. We assume that all amoebots share both. This is not a restrictive assumption because a common compass and chirality can be established efficiently using circuits [12]. The amoebots are controlled in a distributed fashion by identical and anonymous finite state machines. In particular, the amount of memory per amoebot is *constant*, i.e., fixed by the algorithm/state machine running on each amoebot and independent of the number of amoebots in the structure. This means that, for example, unique identifiers for all amoebots cannot be stored. A computation proceeds in *fully synchronous rounds*. In each round, all amoebots act and change their states simultaneously based on their current state and their received signals (see Section 1.2). We measure the time complexity of an algorithm by the number of rounds it requires until all amoebots reach a terminal state. Although the model allows amoebots to perform movements, we only consider static amoebot structures in this paper.



(a) Amoebot structure.

(b) Grid axes and cardinal directions.

(c) Amoebot structure with one global circuit.

(d) Amoebot structure with six circuits.

Figure 1 (a) shows an amoebot structure in the triangular grid. Amoebots are represented by black hexagons and neighboring amoebots are connected by thick edges. (b) shows the axes and cardinal directions in the triangular grid and the unit vector in the East direction. (c, d) illustrate the reconfigurable circuit extension for $c_\phi = 2$. Amoebots are drawn as hexagons, pins are black circles on their borders, and partition sets are drawn as black circles inside the hexagons. The partition sets are connected to the pins they contain. Partition sets in the same circuit have lines of the same color.

1.2 Reconfigurable Circuit Extension

The *reconfigurable circuit extension* [12] places c_ϕ *external links* on every edge connecting two adjacent amoebots $u, v \in A$. The endpoints of an external link are called *pins*. For each link, one pin is owned by u and one is owned by v. The constant c_ϕ is an algorithm design parameter and is the same for all amoebots. In this paper, we use $c_\phi = 2$, which is the least number of pins required by the PASC algorithm [12], a central primitive for our results.

Let $P(u)$ be the set of pins owned by amoebot $u \in A$. Each amoebot u computes a partitioning of $P(u)$ into a set $\mathcal{Q}(u)$ of non-empty, pairwise disjoint subsets $Q \subseteq P(u)$ such that $\bigcup_{Q \in \mathcal{Q}(u)} Q = P(u)$. The subsets $Q \in \mathcal{Q}(u)$ are called *partition sets* and $\mathcal{Q}(u)$ is called the *pin configuration* of u. The amoebot's algorithm defines how the pin configuration is computed in every round. Let $\mathcal{Q} := \bigcup_{u \in A} \mathcal{Q}(u)$ be the set of all partition sets in the amoebot structure. Two partition sets $Q \in \mathcal{Q}(u)$ and $Q' \in \mathcal{Q}(v)$ of neighboring amoebots u and v are *connected* if there is an external link with one pin in Q and one pin in Q'. Let $E_\mathcal{Q}$ be the set of these connections. We call each connected component C of the undirected graph $G_\mathcal{Q} := (\mathcal{Q}, E_\mathcal{Q})$ a *circuit* (see Figs. 1c, d). An amoebot u is part of a circuit C if C contains at least one partition set of u. Note that multiple partition sets of an amoebot u may be contained in the same circuit without u being aware of this due to its lack of global information.

During its activation, each amoebot can establish an arbitrary new pin configuration and send primitive signals called *beeps* on any selection of its partition sets. A beep is broadcast to the circuit containing the partition set it was sent on. All partition sets in that circuit

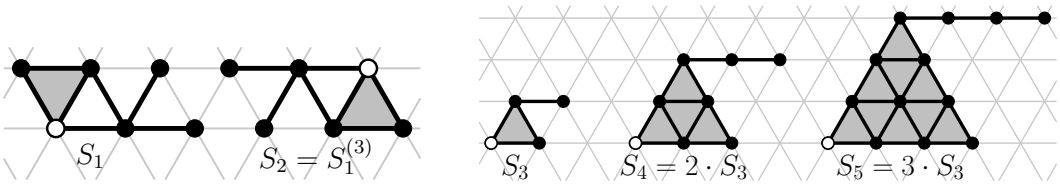


Figure 2 Examples of shapes and shape transformations. Each shape is identified by the grid nodes, edges, and faces it contains. The origin of each shape is highlighted in white (we translate some shapes for better visibility). S_1 and S_2 are equivalent, and each contains one face and one hole. The shapes S_3, S_4 and S_5 illustrate the scaling operation.

receive the beep at the beginning of the next round. An amoebot can tell which of its partition sets have received a beep, but it has no information on the identity, location, or number of beeping amoebots. We model all communication between amoebots with circuits.

1.3 Problem Statement and Our Contribution

Consider the embedding of G_Δ into $\mathbb{R}^2$ such that the grid's faces form unit triangles and one grid node is placed at the plane's *origin* $(0,0) \in \mathbb{R}^2$ (see Fig. 1b). We obtain three coordinate axes, X, Y, Z, and six *cardinal directions* $\mathcal{D} = \{\mathrm{E}, \mathrm{NE}, \mathrm{NW}, \mathrm{W}, \mathrm{SW}, \mathrm{SE}\}$. Let $\boldsymbol{u}_d$ denote the unit vector in direction $d \in \mathcal{D}$.

A *shape* $S \subset \mathbb{R}^2$ is a finite union of nodes, edges, and faces of the embedded grid graph (see Fig. 2, c.f. [10,12]). Edges contain their endpoints and faces contain their enclosing edges. A shape must be connected, but it may contain *holes*, i.e., $\mathbb{R}^2 \setminus S$ might not be connected. This shape definition matches the one used in [10] for shape formation and extends the definition used in [12] for shape recognition.

We define *translation*, *rotation*, and *scaling* operations on a shape S as follows. For $t \in \mathbb{R}^2$, we denote S translated by t by $S + t := \{p + t \mid p \in S\}$. This is a valid shape if and only if t is the position of a grid node, i.e., a linear combination of unit vectors in the cardinal directions with integer coefficients. We can therefore write $t \in V_\Delta$, identifying the position with the unique grid node occupying it. For $r \in \mathbb{Z}$, let $S^{(r)}$ denote S after r counter-clockwise rotations by 60° around the origin of $\mathbb{R}^2$. Note that $r \in \{0, \dots, 5\}$ is sufficient to represent all distinct rotations of a shape in the grid. Let $k \in \mathbb{R}$ be a *scale factor*, then we define $k \cdot S := \{k \cdot p \mid p \in S\}$ to be the shape S scaled by k. We only consider positive integer scale factors to ensure that the resulting set is a valid shape. If S is *minimal*, i.e., there is no scale factor $0 < k' < 1$ such that $k' \cdot S$ is a valid shape, then the integer scale factors cover all possible scales of S that produce valid shapes (see Lemma 1 in [10]). Two shapes are *equivalent* if one can be obtained from the other by a rigid motion, i.e., a composition of a translation and a rotation.

Let $V(S) \subset V_\Delta$ denote the set of grid nodes contained in S. For convenience, we assume $(0,0) \in V(S)$ for any given shape S and call this node the *origin* of S. This property is invariant under rotation and scaling as defined above. A *valid placement* of a shape S in an amoebot structure A is an amoebot $p \in A$ with $V(S + p) \subseteq A$ (identifying p with the grid node it occupies). Let $\mathcal{V}(S, A) \subseteq A$ denote the set of all valid placements of S in A. The *maximum scale* of S in A is the largest scale $k \in \mathbb{N}_0$ such that there is a valid placement of $k \cdot S^{(r)}$ in A for some $r \in \mathbb{Z}$:

$$k_{\max}(S, A) := \sup \left\{ k \in \mathbb{N}_0 \;\middle|\; \exists\, r \in \mathbb{Z} : \mathcal{V}\left(k \cdot S^{(r)}, A\right) \neq \emptyset \right\}$$

$k_{\max}$ is well-defined because $0 \cdot S$ is at most a single node for every shape S, which fits into

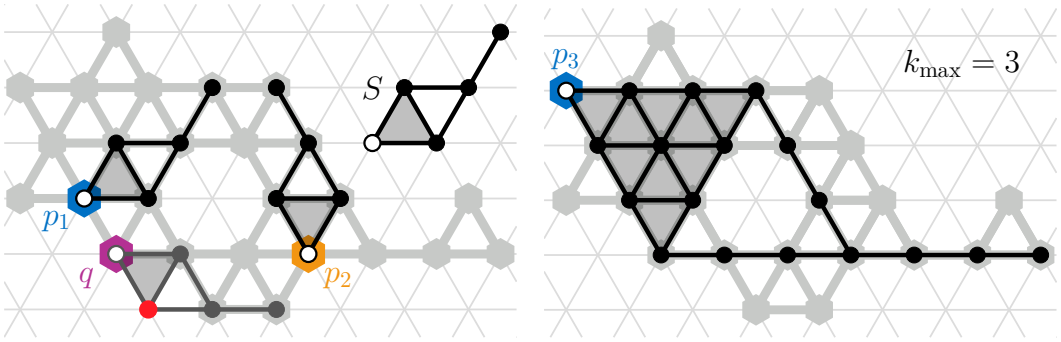


Figure 3 Valid placements of a shape S in the amoebot structure A from Fig. 1a. p_1 and p_2 are valid placements of S and $S^{(1)}$, respectively. q is not a valid placement of $S^{(5)}$ because one node of $V(S^{(5)} + q)$ is not contained in A. p_3 is the only valid placement of $3 \cdot S^{(5)}$ and there are no valid placements for any scale $k > 3$, so we have $k_{\max}(S, A) = 3$. A shape containment algorithm allows p_3 to determine $p_3 \in \mathcal{V}(k_{\max} \cdot S^{(5)}, A)$ and rules out all other amoebots and rotations.

any non-empty amoebot structure A. We obtain $k_{\max}(S, A) = \infty$ if and only if S does not contain any edges. If S and A are clear from the context, we will write $k_{\max} = k_{\max}(S, A)$. See Fig. 3 for illustration.

We define the *shape containment problem* as follows: Let S be a shape (containing the origin). An algorithm solves the shape containment problem instance (S, A) for an amoebot structure A if it terminates eventually and at the end,

1. all amoebots know whether $k_{\max} \in \{0, \infty\}$ (meaning that all or none of the amoebots are valid placements), and
2. if $k_{\max} \in \mathbb{N}$, then for every $r \in \{0, \dots, 5\}$, each amoebot knows whether it is contained in $\mathcal{V}(k_{\max} \cdot S^{(r)}, A)$.

The algorithm solves the shape containment problem for S if it solves the shape containment instances (S, A) for all finite connected amoebot structures A. Note that the shape S is fixed for an algorithm, i.e., it is encoded in the state machine in some way, and its size is constant. The algorithm may use an equivalent shape S' instead of S itself.

There are two key challenges in solving the shape containment problem. First, the amoebots have to find the maximum scale $k_{\max}$. We approach this problem by testing individual scale factors for valid placements until $k_{\max}$ is fixed. We call this part of an algorithm the *scale factor search*. Second, for a given scale k and a rotation r, the valid placements of $k \cdot S^{(r)}$ have to be identified. This means that every amoebot $p \in A$ must collect information on all amoebots in $V(k \cdot S^{(r)} + p)$. p then has to mark itself as a valid placement if and only if all of these amoebots exist. In some cases, we initially view all amoebots as *placement candidates* and then *eliminate* candidates that can be ruled out as valid placements. We call this part the *valid placement search* procedure. Since the information about missing amoebots naturally comes from the boundaries of the structure, the main difficulty here is distributing this information efficiently to all affected amoebots and not ruling out any valid placements.

In this paper, we present sublinear time solutions for the shape containment problem using circuits. As a motivation, we first prove a lower bound for the general case that holds even if the maximum scale is known, by creating a communication bottleneck for an example shape. Next, we present scale factor search approaches using distributed memory, applying a binary search where it is possible. We then introduce simple and efficient primitives for placement search algorithms based on combining and transforming valid placements of lines, thereby

constructing increasingly complex shapes. Using earlier results for reconfigurable circuits, these primitives run in at most logarithmic time. By combining our primitives, we obtain the class of *snowflake* shapes, which contains a variety of complex and practically relevant shapes. For example, it contains all convex shapes, all shapes that are composed of parallelograms of the same orientation, all shapes composed of hexagons, and combinations (e.g., unions) of these and other shapes. The definition of this class allows more types of shapes to be combined and integrated when a valid placement search procedure is given for them. Our shape containment solution for snowflakes takes a sublinear number of rounds. Further, we show that for the set of *star convex* shapes, which is contained in the snowflake class, a binary search for the scale factor even leads to a polylogarithmic solution. Surprisingly, the binary search approach is *only* directly applicable to star convex shapes. We only give high-level explanations of our primitives and defer the technical details and proofs to the full version of the paper [4]. Selected proofs can also be found in the appendix.

1.4 Related Work

The authors of [12] demonstrated the potential of their reconfigurable circuit extension with algorithms solving the leader election, compass alignment, and chirality agreement problems within $\mathcal{O}(\log n)$ rounds, w.h.p.[1] They also presented efficient solutions for some exact shape recognition problems: Given common chirality, an amoebot structure can determine whether it matches a scaled version of a given shape composed of edge-connected faces in $\mathcal{O}(1)$ rounds. Without common chirality, convex shapes can be detected in $\mathcal{O}(1)$ rounds and parallelograms with linear or polynomial side ratios can be detected in $\Theta(\log n)$ rounds, w.h.p.

The PASC algorithm was introduced in [12] and refined in [21], and it allows amoebots to compute distances along chains. It has become a central primitive in the reconfigurable circuit extension, as it was used to construct spanning trees, detect symmetry, and identify centers and axes of symmetry in polylogarithmic time, w.h.p. [21]. The authors in [20] used it to solve the single- and multi-source shortest path problems, requiring $\mathcal{O}(\log \ell)$ rounds for a single source and ℓ destinations and $\mathcal{O}(\log n \log^2 k)$ rounds for k sources and any number of destinations. The PASC algorithm also plays a crucial role in this paper (see Sec. 2.2.2).

The authors in [11] studied the capabilities of a generalized circuit communication model that directly extends the reconfigurable circuit model to general graphs. They provided polylogarithmic time algorithms for various common graph construction (minimum spanning tree, spanner) and verification problems (minimum spanning tree, cut, Hamiltonian cycle etc.). Additionally, they presented a generic framework for translating a type of lower bound proof from the widely used CONGEST model into the circuit model, demonstrating that some problems are hard in both models, while others can be solved much faster with circuits. For example, checking whether a graph contains a 5-cycle takes $\Omega(n/\log n)$ rounds in general graphs, even with circuits, while the verification of a connected spanning subgraph can be done with circuits in $\mathcal{O}(\log n)$ rounds w.h.p., which is below the lower bound shown in [22].

In the context of computational geometry, the basic polygon containment problem was studied in [6], focusing on the case where only translation and rotation are allowed. The problem of finding the largest copy of a convex polygon inside some other polygon was discussed in [23] and [1], for example. An example for the problem of placing multiple polygons inside another without any polygons intersecting each other is given by [17]. More

[1] An event holds *with high probability* (w.h.p.) if it holds with probability at least $1 - n^{-\delta}$, where the constant δ can be made arbitrarily large.

recently, the authors in [16] showed lower bounds for several polygon placement cases under the kSUM conjecture. For example, assuming the 5SUM conjecture, there is no $\mathcal{O}((p+q)^{3-\varepsilon})$-time algorithm for any $\varepsilon > 0$ that finds a largest copy of a simple polygon P with p vertices that fits into a simple polygon Q with q vertices under translation and rotation. Perhaps more closely related to our setting (albeit centralized) is an algorithm that solves the problem of finding the largest area parallelogram inside of an object in the triangular grid, where the object is a set of edge-connected faces [2].

2 Preliminaries

This section introduces elementary algorithms for the circuit extension from previous work.

2.1 Coordination and Synchronization

As mentioned before, we assume that all amoebots share a common compass direction and chirality. This is a reasonable assumption because the authors of [12] have presented randomized algorithms establishing both in $\mathcal{O}(\log n)$ rounds, w.h.p.

We often want to synchronize amoebots, for example, when different parts of the structure run independent instances of an algorithm simultaneously. For this, we can use a *global circuit*: Each amoebot connects all of its pins into a single partition set. The resulting circuit spans the whole structure and allows the amoebots which are not yet finished with their procedure to inform all other amoebots by sending a beep. When no beep is sent, all amoebots know that all instances of the procedure are finished. Due to the fully synchronous scheduler, we can establish the global circuit periodically at predetermined intervals.

2.2 Chains and Chain Primitives

A *chain* of amoebots with length $m-1$ is a sequence of m amoebots $C = (p_0, \dots, p_{m-1})$ where all subsequent pairs p_i, p_{i+1}, $0 \leq i < m-1$, are neighbors, each amoebot except p_0 knows its predecessor, and each amoebot except p_{m-1} knows its successor. In this paper, we only consider *simple* chains without multiple occurrences of the same amoebot. By letting each amoebot decide whether it connects a pin toward its predecessor with a pin toward its successor, we can easily establish circuits along such chains.

2.2.1 Binary Operations

The constant memory limitation of amoebots makes it difficult to deal with non-constant data, particularly numbers that can increase with n. However, we can use amoebot chains to implement a distributed memory by letting each amoebot on the chain store one bit of a binary number, as demonstrated in [7, 21]. Using circuits, we can implement efficient comparisons and arithmetic operations between two operands stored on the same chain.

▶ **Lemma 1.** *Let $C = (p_0, \dots, p_{m-1})$ be an amoebot chain such that each amoebot p_i stores two bits a_i and b_i of the integers a and b, where $a = \sum_{i=0}^{m-1} a_i 2^i$ and $b = \sum_{i=0}^{m-1} b_i 2^i$. Within $\mathcal{O}(1)$ rounds, the amoebots on C can compare a to b and compute the first m bits of $a+b$, $a-b$ (if $a \geq b$), $2 \cdot a$, and $\lfloor a/2 \rfloor$ and store them on the chain. Within $\mathcal{O}(m)$ rounds, the amoebots on C can compute the first m bits of $a \cdot b$, $\lfloor a/b \rfloor$, and $a \bmod b$ and store them on the chain.*

Lemma 1 is a minor improvement over the algorithms presented in [21]. Additionally, individual amoebots can execute simple binary operations online on *streams* of bits:

▶ **Lemma 2.** *Let p be an amoebot that receives two numbers a, b as* bit streams*, i.e., it receives the bits a_i and b_i in the i-th iteration of some procedure, for $i = 0, \dots, m$. Then, p can compute bit c_i of $c = a + b$ or $c = a - b$ (if $a \geq b$) in the i-th iteration and determine the comparison result between a and b by iteration m, with only constant overhead per iteration.*

2.2.2 The PASC Algorithm

A particularly useful algorithm in the reconfigurable circuit extension is the *Primary-And-Secondary-Circuit* (PASC) algorithm, first introduced in [12]. We omit the details of the algorithm and only outline its relevant properties. Please refer to [21] for details.

▶ **Lemma 3** ([12, 21]). *Let $C = (p_0, \dots, p_{m-1})$ be a chain of m amoebots. The PASC algorithm, executed on C with start point p_0, performs $\lceil \log m \rceil$ iterations within $\mathcal{O}(\log m)$ rounds. In iteration $j = 0, \dots, \lceil \log m \rceil - 1$, each amoebot p_i computes the j-th bit of its distance i to the start of the chain, i.e., p_i computes i as a bit stream.*

The PASC algorithm is especially useful with binary counters. It allows us to compute the length of a chain, which is received by the last amoebot in the chain and can be stored in binary on the chain itself (letting the last amoebot send the received bits as beeps on a circuit spanning the chain). Furthermore, given some binary counter storing a distance ℓ and some amoebot chain $C = (p_0, \dots, p_{m-1})$, each amoebot p_i can compare i to ℓ by receiving the bits of ℓ on a global circuit in sync with the iterations of the PASC algorithm on C.

▶ **Lemma 4.** *Let $C = (p_0, \dots, p_{m-1})$ be a chain in an amoebot structure A and let a value $\ell \in \mathbb{N}_0$ be stored in some binary counter of A. Within $\mathcal{O}(\log \min\{\ell, m\})$ rounds, every amoebot p_i can compare i to ℓ. The procedure can run simultaneously on any set of edge-disjoint chains with length $\leq m - 1$.*

3 A Lower Bound for Finding Valid Placements

We first show a lower bound that demonstrates a central difficulty arising in the shape containment problem. For a simple example shape S_Ω (see Fig. 4), we show that even if the maximum scale is known, identifying all valid placements of S_Ω can require $\Omega(\sqrt{n})$ rounds due to communication bottlenecks.

▶ **Theorem 5.** *There exists a shape S_Ω such that for any choice of origin and every amoebot algorithm $\mathcal{A}$ that terminates after $o(\sqrt{n})$ rounds, there exists an amoebot structure A for which the algorithm does not compute $\mathcal{V}(k_{\max}(S_\Omega, A) \cdot S_\Omega, A)$, even if $k_{\max}$ is known.*

The proof idea is as follows (see Appendix A for details): Consider an amoebot structure A consisting of two parts, X and Y, which are only connected by a single edge. In one round, one of at most 2^{c_ϕ} different signals can be sent via the c_ϕ external links on that edge. Suppose that for $k = k_{\max}(S_\Omega, A)$, all valid placements of $k \cdot S_\Omega$ lie in X, and each placement's required node set reaches into Y. By adding and removing amoebots from Y, we can control which amoebots in X are valid placements. The idea for our lower bound proof is to construct 2^k distinct patterns of valid placements by only altering Y. To identify the valid placements correctly, the amoebots in X must distinguish between all of these patterns using information that must cross the edge between X and Y. Since c_ϕ is a constant, this takes $\Omega(k)$ rounds. We show such a construction for any $k \geq 2$, with $n = \mathcal{O}(k^2)$, and for any choice of origin in S_Ω. In the remainder of this paper, we explore for which shapes these arbitrary patterns of valid placements do not occur, and present efficient procedures exploiting this property.

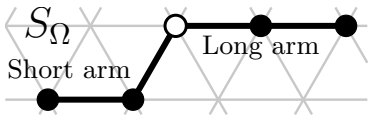


Figure 4 An example shape for which the valid placement search is bounded below by $\Omega(\sqrt{n})$.

This lower bound holds for the problem of finding valid placements. Since every algorithm solving the shape containment problem must compute all valid placements for the maximum scale, the bound also holds for the general shape containment problem. Further, note that the lower bound of $\Omega(D)$ in the amoebot model without circuits, where $D = \Omega(\sqrt{n})$ is the diameter of A, is caused by the *distance* information can travel per time step. In contrast, this lower bound is caused by the limited *bandwidth* of the circuits in some amoebot structures.

4 Scale Factor Search

As outlined before, our shape containment algorithms consist of two search procedures. The first is a *scale factor search* that determines which scales have to be checked to find the maximum scale. The second procedure is a *valid placement search* that identifies all valid placements of $k \cdot S^{(r)}$ for all $r \in \{0, \dots, 5\}$, given the scale k in a binary counter.

Consider some shape S and an amoebot structure A with a binary counter storing an upper bound $K \geq k_{\max}(S, A)$. The simple *linear search* procedure runs valid placement checks for the scales $K, K-1, \dots, 1$ and accepts when the first valid placement is found. If no placement is found in any iteration, we have $k_{\max} = 0$.

▶ **Lemma 6.** *Let S be a shape and A an amoebot structure with a binary counter storing an upper bound $K \geq k_{\max}(S, A)$. Given a valid placement search procedure for S, the amoebots compute $k_{\max}(S, A)$ in at most K iterations, running the placement search for scales $K, K-1, \dots, k_{\max}$ and with constant overhead per iteration.*

When using the linear search method, finding a small upper bound K is essential for reducing the runtime. One way of obtaining K for S is to use a simpler shape S' with $S' \subseteq S$ for which $k_{\max}$ can be computed efficiently. Since every valid placement of S is also a valid placement of S', we have $k_{\max}(S, A) \leq k_{\max}(S', A) =: K$. Moving on, some shapes permit a faster search method based on an inclusion relation between different scales.

▶ **Definition 7.** *We call a shape S* self-contained *if for all scales $k < k'$, there exist a translation $t \in \mathbb{R}^2$ and a rotation $r \in \{0, \dots, 5\}$ such that $k \cdot S^{(r)} + t \subseteq k' \cdot S$.*

For self-contained shapes, finding no valid placements at scale k immediately implies $k_{\max}(S, A) < k$, which allows us to apply a *binary search*: Starting with scale $k = 1$, we double the scale factor until no valid placements are found at some scale $K = 2^j$, which we then use as the upper bound for a binary search between 1 and K.

▶ **Lemma 8.** *Let S be a self-contained shape and let A be an amoebot structure with a binary counter large enough to store $k_{\max} = k_{\max}(S, A)$. Given a valid placement search procedure for S, the amoebots can compute $k_{\max}$ within $\mathcal{O}(\log k_{\max})$ iterations such that each iteration runs the valid placement search once for some scale $k \leq 2 \cdot k_{\max}$ and has constant overhead.*

5 Efficient Placement Search Procedures

In this section, we introduce placement search procedures allowing amoebots to identify valid placements of shapes when their scale is given in a binary counter. To cover all rotations, we simply repeat the placement search six times with rotated directions. We only provide high-level descriptions of the algorithms. Please refer to the full paper [4] for details. Our procedures heavily rely on combining the PASC algorithm with binary operations on bit streams (see Lemma 4). A simple and natural way to establish the required chains is using *segments*:

▶ **Definition 9.** *Let $W \in \{X, Y, Z\}$ be a grid axis. A* (W)-segment *is a connected set of grid nodes on a line parallel to W. Let $C \subseteq V_\Delta$, then a* maximal W-segment of C *is a finite W-segment $M \subseteq C$ that cannot be extended with nodes from C on either end. The* length *of a finite segment M is $|M| - 1$.*

For example, chains on maximal segments of the amoebot structure A can be constructed easily once a direction has been agreed upon: All amoebots on a segment identify their chain predecessor and successor by checking the existence of neighbors on the direction's axis. The start and end points of the segment are the unique amoebots lacking a neighbor in one or both directions. Using chains on segments, we can find valid placements of *line shapes*.

▶ **Definition 10.** *A* line shape $\mathrm{L}(d, \ell)$ *is a shape consisting of $\ell \in \mathbb{N}_0$ consecutive edges extending in direction d from the origin. For $\ell = 0$, the shape only contains the origin point.*

Lines are the fundamental primitive shapes we will use to construct much more complex shapes. Our placement search procedure for lines runs the PASC algorithm to compute indices along maximal amoebot segments and compares them to the given line length ℓ, which is transmitted on a global circuit. Exactly the amoebots whose chain index, i.e., distance to the end of the segment, is at least ℓ are valid placements of the line. If the line shape L has length ℓ and the current scale is k, we compute $\ell' = k \cdot \ell$ in the counter storing k and run the procedure for length ℓ'. Note that for line shapes, the boundary amoebots holding the information which placements are invalid lie on the same segment as the affected placement candidates, enabling this simple solution.

▶ **Lemma 11.** *Let $L = \mathrm{L}(d, \ell)$ be a line shape and let A be an amoebot structure that stores ℓ in some binary counter and where all amoebots agree on d. Within $\mathcal{O}(\log \min\{\ell, n\})$ rounds, the amoebots can compute $\mathcal{V}(L, A)$.*

Recall that for any two shapes S, S' with $S \subseteq S'$, $k_{\max}(S, A)$ is an upper bound for $k_{\max}(S', A)$. Thus, the maximum scale of an edge $\mathrm{L}(d, 1)$ is a natural upper bound on the maximum scale of any non-trivial shape. In particular, any longest segment in the amoebot structure provides sufficient memory to store the scale values that must be considered. To use this fact, we establish binary counters on *all* amoebot segments (on all axes) and use them simultaneously, deactivating counters if they run out of memory.

We now discuss efficient operations on valid placements that allow us to determine the valid placements of a transformed shape. The first, simple operation is the *union* of shapes. Given the valid placements $C_1 = \mathcal{V}(S_1, A)$ and $C_2 = \mathcal{V}(S_2, A)$ of two shapes S_1 and S_2, the amoebots in A can find the valid placements of $S' = S_1 \cup S_2$ in a single round: Due to the relation $\mathcal{V}(S', A) = C_1 \cap C_2$, each amoebot only has to check whether it is a valid placement of both shapes. Observe that S' is always connected because we assume that S_1 and S_2 contain the origin, and the location of the origin in S_1 and S_2 directly affects the shape S'.

5.1 The Minkowski Sum Primitive

Next, we consider the *Minkowski sum* of a shape with a line.

▶ **Definition 12.** *Let S_1, S_2 be two shapes, then their* Minkowski sum *is defined as*

$$S_1 \oplus S_2 := \{p_1 + p_2 \mid p_1 \in S_1, p_2 \in S_2\}.$$

The resulting subset of $\mathbb{R}^2$ is a valid shape, and if both shapes contain the origin, then their sum also contains the origin. Observe that for any shape S and any line $\mathrm{L}(d, \ell)$, we have

$$V(S \oplus \mathrm{L}(d, \ell)) = \bigcup_{i=0}^{\ell} V(S + i \cdot \boldsymbol{u}_d),$$

i.e., $V(S \oplus \mathrm{L}(d, \ell))$ consists of $\ell + 1$ consecutive copies of $V(S)$. This means that an amoebot p is a valid placement of $S' = S \oplus \mathrm{L}(d, \ell)$ if and only if it is a valid placement of the line $L = \mathrm{L}(d, \ell)$ and for every amoebot $q \in V(L + p)$, q is a valid placement of S. Suppose the amoebots already know the set C of valid placements of S and store ℓ in a binary counter. Then, they can find the valid placements of $S \oplus L$ by running the placement search for L within C, treating the amoebots in $A \setminus C$ as if they did not exist, except for synchronization.

▶ **Lemma 13.** *Let S be an arbitrary shape, $L = \mathrm{L}(d, \ell)$ a line, and A an amoebot structure storing ℓ and a scale k in some binary counter. Given d and the set $C = \mathcal{V}(k \cdot S, A)$, the amoebots can compute $\mathcal{V}(k \cdot (S \oplus L), A)$ within $\mathcal{O}(\log \min\{k \cdot \ell, n\})$ rounds.*

Observe that this already yields efficient valid placement search procedures for shapes like parallelograms $\mathrm{L}(d_1, \ell_1) \oplus \mathrm{L}(d_2, \ell_2)$ and unions thereof.

5.2 The Translation Primitive

Consider some shape S, its valid placements $C = \mathcal{V}(S, A)$, a direction $d \in \mathcal{D}$, and a distance $\ell \in \mathbb{N}$ such that $\ell \cdot \boldsymbol{u}_d \in S$. Now, let $S' = S - \ell \cdot \boldsymbol{u}_d$, then S' only differs from S in the location of its origin, and the valid placements of S' are just translated versions of the placements of S, i.e., $\mathcal{V}(S', A) = C + \ell \cdot \boldsymbol{u}_d$. Given the valid placements of S, we can compute the placements of S' with a procedure that translates the placement information from C. Our *translation primitive* can do this efficiently for shapes with the following property:

▶ **Definition 14.** *Let S be a shape and $W \in \{X, Y, Z\}$ a grid axis. The* minimal axis width of S on W *(or W-*width*) is the infimum of the length of any maximal line segment in S that is parallel to W. We call S (W-)*wide *or* wide on W *if its W-width is at least* 1.

For example, the minimal axis width of a triangular face is 0 for all axes due to its corners, and the W-width of $\mathrm{L}(d, \ell)$ is ℓ when d is parallel to W and 0 otherwise. Further, observe that for any shape S, direction d on axis W, and $\ell \in \mathbb{N}$, the W-width of $S \oplus \mathrm{L}(d, \ell)$ is at least ℓ.

Now, consider a W-wide shape S and let $C = \mathcal{V}(k \cdot S, A)$ for some scale $k \in \mathbb{N}$. Note that $k \cdot S$ has a W-width of at least k. Let $q \in A \setminus C$ be an *invalid* placement of $k \cdot S$, then q must be part of a W-segment of $A \setminus C$ that has length at least k or is bounded by unoccupied nodes on at least one side. This is because any unoccupied node $x \in V_\Delta \setminus A$ that makes q invalid lies in $V(k \cdot S + q)$. Since $V(k \cdot S)$ consists of W-segments of length at least k, x causes a similar segment which contains q to be invalid, as any placement in this segment is also invalidated by x.

For translating the valid placements C of $k \cdot S$ by k steps in a direction d parallel to W, suppose that $\mathrm{L}(d,1) \subseteq S$ (otherwise, we add this edge as part of the shape transformation). Then, we can reduce the problem to translating a single segment of length at least k. For this, we run a procedure in every maximal W-segment M of A simultaneously. Suppose we want to translate some segment C_M in M. It suffices to translate the start p_1 and endpoint p_2 of C_M, identifying $p_1' = p_1 + k \cdot \boldsymbol{u}_d$ and $p_2' = p_2 + k \cdot \boldsymbol{u}_d$. To identify p_i', the amoebots in M run the PASC algorithm with p_i as the start point while transmitting k on a global circuit. Each amoebot compares its distance to p_i to k, and the amoebot with distance equal to k becomes p_i'. Afterward, the amoebots in M establish a chain circuit between p_1' and p_2', and send a beep that is received by all amoebots on the translated segment. Due to the size of the considered segments C_M, this procedure can run for almost all segments within M simultaneously without interference. The translation primitive moves the origin of a W-wide shape on the axis W and adds the line connecting the old and new origin locations.

▶ **Lemma 15.** *Let S be a W-wide shape and A an amoebot structure that stores a scale k in a binary counter and knows $\mathcal{V}(k \cdot S, A)$. Given a direction d on axis W and $\ell \in \mathbb{N}$, the amoebots can compute $\mathcal{V}(k \cdot ((S + \ell \cdot \boldsymbol{u}_d) \cup \mathrm{L}(d,\ell)), A)$ within $\mathcal{O}(\ell \cdot \log \min\{k \cdot \ell, n\})$ rounds.*

5.3 The Triangle Primitive

Finally, we show how to find the valid placements of *triangles* using the above primitives. Line shapes are scaled edges and triangles are scaled faces.

▶ **Definition 16.** *Let $\mathrm{T}(d,1)$ be the shape consisting of the single triangular face spanned by the edges $\mathrm{L}(d,1)$ and $\mathrm{L}(d',1) = \mathrm{L}(d,1)^{(1)}$. We define general* triangle shapes *as $\mathrm{T}(d,\ell) := \ell \cdot \mathrm{T}(d,1)$ for $\ell \in \mathbb{N}_{>1}$ and call ℓ the* side length *or* size *of $\mathrm{T}(d,\ell)$.*

To construct the valid placements of $T = \mathrm{T}(d,\ell)$, we cover $V(T)$ with the union of three parallelograms. The side lengths of the parallelograms are $\ell' = \lfloor \ell/2 \rfloor$ and $\ell'' = \lfloor \ell/2 \rfloor + (\ell \bmod 2)$. Two parallelograms are translated to reach the corners of the triangle. The algorithm can be summarized as follows: The amoebots first compute ℓ' and ℓ'' on the binary counter storing ℓ. Then, they run the line primitive to compute the valid placements of three lines of length ℓ' or ℓ''. Next, they apply the Minkowski sum primitive to the three sets of line placements, resulting in the valid placements of three parallelograms. To translate two of the parallelograms, they run the translation primitive, which is applicable since the parallelograms have a width of at least ℓ' on the translation axis (for a translation by $\ell'' = \ell' + 1$, the valid placements can be moved by one more step in a single round using only local communication). Finally, they intersect the valid placements of all three parallelograms (union operation) to obtain the valid placements of the triangle.

▶ **Lemma 17.** *Let $T = \mathrm{T}(d,\ell)$ be a triangle shape and let A be an amoebot structure that stores ℓ in some binary counter. The amoebots can compute $\mathcal{V}(T, A)$ within $\mathcal{O}(\log \min\{\ell, n\})$ rounds.*

6 Shape Classification

We now generalize the idea of the triangle primitive to define the class of *snowflake* shapes, whose valid placements can be found efficiently using our primitives. Then, we discuss interesting subsets of these shapes and combine our valid placement search procedures with suitable scale factor search approaches to solve the shape containment problem.

6.1 Snowflake Shapes

Combining the primitives discussed in the previous section, we obtain the following class of shapes. Our definition identifies shapes with trees such that every node in the tree represents a shape and every edge represents a composition or transformation of shapes.

▶ **Definition 18.** *A* snowflake tree *is a finite, non-empty tree $T = (V_T, E_T)$ with three node labeling functions $\tau : V_T \to \{\mathrm{L}, \mathrm{T}, \cup, \oplus, +\}$, $d : V_T \to \mathcal{D}$ and $\ell : V_T \to \mathbb{N}_0$ that satisfies the following constraints. Every node $v \in V_T$ represents a shape S_v such that:*

- *If $\tau(v) = \mathrm{L}$, then v is a leaf node and $S_v = \mathrm{L}(d(v), \ell(v))$* (line node).
- *If $\tau(v) = \mathrm{T}$, then v is a leaf node and $S_v = \mathrm{T}(d(v), \ell(v))$, where $\ell(v) > 0$* (triangle node).
- *If $\tau(v) = \cup$, then $S_v = \bigcup_{i=1}^{m} S_{u_i}$, where $u_1, \dots, u_m$ are the children of v and $m \geq 2$* (union node).
- *If $\tau(v) = \oplus$, then $S_v = S_u \oplus \mathrm{L}(d(v), \ell(v))$, where u is the unique child of v and $\ell(v) > 0$* (sum node).
- *If $\tau(v) = +$, then $S_v = (S_u + \ell(v) \cdot \boldsymbol{u}_{d(v)}) \cup \mathrm{L}(d(v), \ell(v))$, where u is the unique child of v, S_u has a minimal axis width > 0 on the axis of $d(v)$, and $\ell(v) > 0$* (translation node).

Let $r \in V_T$ be the root of T, then we say that S_r is the snowflake shape *represented by T.*

Note that this definition constrains the location of a snowflake's origin. Because algorithms for the shape containment problem can place the origin of a shape freely, we may include equivalent shapes in the class of snowflakes. Our algorithms always place the origin by the definition. Fig. 5 shows examples for snowflakes and non-snowflake shapes.

To compute the valid placements of a snowflake, we apply the primitives from Section 5 following a *topological ordering* of the shape's tree representation. This way, the amoebots first compute valid placements of primitive shapes (lines and triangles), and then they apply the union, Minkowski sum, and translation operations successively until they arrive at the valid placements of the shape represented by the root node. This sequence of operations is encoded in the state machine of each amoebot; we say that the amoebots "have access" to the shape and its tree representation.

Consider a snowflake S represented by a tree $T = (V_T, E_T)$ with labelings $\tau(\cdot)$, $d(\cdot)$ and $\ell(\cdot)$. We assume that each amoebot has access to a representation of T and a topological ordering of T from the leaves to the root. The amoebots compute $\mathcal{V}(k \cdot S, A)$ as follows:

1. For every leaf $v \in V_T$, perform the placement search for the scaled primitive $\mathrm{L}(d(v), k \cdot \ell(v))$ or $\mathrm{T}(d(v), k \cdot \ell(v))$ represented by v. Let $C(v) \subseteq A$ be the resulting set of valid placements.
2. Process each non-leaf node $v \in V_T$ in the topological ordering as follows:
   a. If $\tau(v) = \cup$, then set $C(v) = \bigcap_{i=1}^{m} C(u_i)$, where the u_i are the child nodes of v.
   b. If $\tau(v) = \oplus$, let u be the unique child of v. Run the procedure from Section 5.1 to compute $C(v)$ from $C(u)$.
   c. If $\tau(v) = +$, let u be the unique child of v. Run the procedure from Section 5.2 to compute $C(v)$ from $C(u)$.
3. Let $r \in V_T$ be the root of T. Terminate with success ($k_{\max} \in \mathbb{N}$) and report the valid placements as $\mathcal{V}(k \cdot S, A) = C(r)$ if $C(r) \neq \emptyset$, otherwise terminate with failure ($k_{\max} = 0$).

▶ **Lemma 19.** *Let A be an amoebot structure storing a scale k in some binary counter and let S be a snowflake. Given an encoding of the tree T of S with a topological ordering, the amoebots can compute $\mathcal{V}(k \cdot S^{(r)}, A)$ for $r \in \{0, \dots, 5\}$ within $\mathcal{O}(\log \min\{k, n\})$ rounds.*

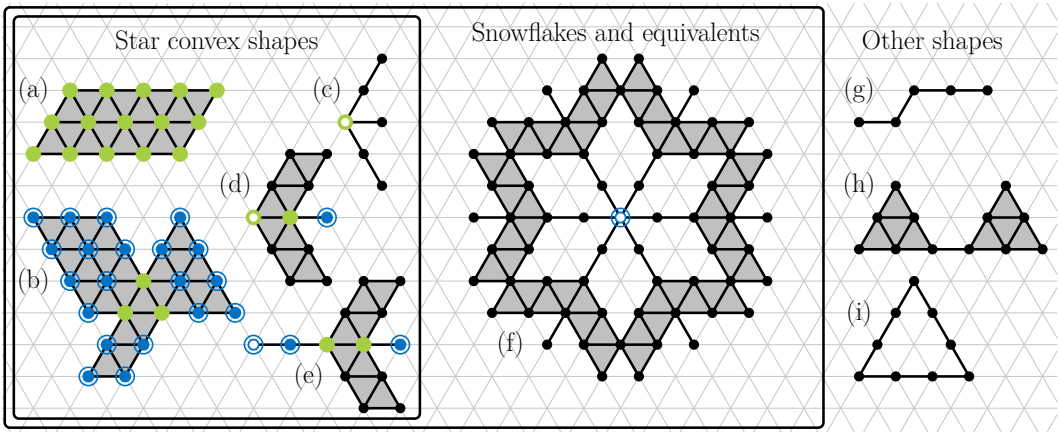


Figure 5 Examples of snowflakes, star convex shapes, and non-snowflake shapes. Green nodes indicate star convex shape centers, encircled blue nodes are possible snowflake origins, and origins used for operations have a white center. All center nodes are also snowflake origins. Shape (a) is convex and shape (b) demonstrates that not all snowflake origins must be center nodes. Shape (c) is the union of three lines, (d) is the Minkowski sum of (c) and $\mathrm{L}(\mathrm{E}, 1)$, shape (e) is the union of $\mathrm{L}(\mathrm{E}, 2)$ and (d) $+ 2 \cdot \boldsymbol{u}_{\mathrm{E}}$, and shape (f) is the union of six rotated variants of shape (e). (g) is the example shape S_Ω for the lower bound from Section 3.

The class of snowflake shapes is the result of our primitive operations and is therefore somewhat artificial. However, it contains a large variety of shapes, and its modular definition provides a framework for constructing efficient valid placement search algorithms for more shapes. For example, any shape that is composed of parallelograms with the same orientation is a snowflake, even if the parallelograms are only connected at the corners. This is because a parallelogram $\mathrm{L}(d_1, \ell_1) \oplus \mathrm{L}(d_2, \ell_2)$ is both W_1-wide and W_2-wide, where W_i is the axis of d_i. Therefore, the translation primitive can be used to move the shape's origin to any position in the parallelogram, after which the union primitive allows the addition of another parallelogram at this position. Since this maintains the width properties, it can be repeated until all parallelograms have been added to the shape. A similar method is possible for shapes composed of hexagons because hexagons are even wide on all three axes. Constructing shapes out of building blocks like parallelograms is a popular approach for shape formation in grid-based modular robot systems [3, 19]. Shapes in square grid models can be represented by parallelograms in the triangular grid. Furthermore, any valid placement search algorithm for non-snowflake shapes can be integrated into this framework as a primitive to expand the set of shapes.

6.2 Star Convex Shapes

To combine our valid placement search procedure with a scale factor search, we would like to know for which shapes the binary search approach is applicable. Recall from Section 4 that at least *self-contained* shapes permit a binary search. In this section, we characterize this class of shapes exactly as the *star convex* shapes.

▶ **Definition 20.** *A shape S is* star convex *if it is hole-free and contains a* center node c *such that for every $v \in V(S)$, all shortest paths from c to v in G_Δ are contained in S.*

For example, all convex shapes are star convex since all their nodes are centers. It is easy to see that star convex shapes are self-contained: When placing the origin of S on a center node, no translation is necessary and $k \cdot S \subseteq k' \cdot S$ holds for all $k, k' \in \mathbb{N}, k < k'$. We can

even show that *only* star convex shapes are self-contained. As the authors of [15] point out in their extensive survey on *starshaped sets* (see p. 1007), the results in [18] show that these two properties are equivalent in much more general settings when rotations are omitted.

▶ **Theorem 21.** *A shape is self-contained if and only if it is star convex.*

The main proof and supporting lemmas can be found in Appendix B. In our context, this equivalence implies that the efficient binary search can only be applied directly to star convex shapes. For any non-star convex shape S, there exist an amoebot structure A and scale factors k, k', $k < k'$, such that $\mathcal{V}(k \cdot S^{(r)}, A) = \emptyset$ for all r but $\mathcal{V}(k' \cdot S^{(r)}, A) \neq \emptyset$ for some r; consider, e.g., $A = V(k' \cdot S)$ for sufficiently large k and k'. The proof of Theorem 21 shows that this always holds for infinitely many scales k, k'.

6.3 Shape Containment Solutions

Finally, we obtain algorithms solving the shape containment problem by combining valid placement search procedures with suitable scale factor search methods. First, we observe that star convex shapes are snowflakes:

▶ **Lemma 22.** *Every star convex shape S is equivalent to a snowflake. If its origin is a center node, S itself is a snowflake.*

This implies that we can solve the shape containment problem for a star convex shape S in just $\mathcal{O}(\log^2 k_{\max}(S, A))$ rounds by using a binary search. For a snowflake S that is not star convex, we apply the linear search approach, which runs $\mathcal{O}(K)$ valid placement searches when given some upper bound $K \geq k_{\max}(S, A)$. To obtain this upper bound, we use the observation that a snowflake without faces will always be a union of line shapes meeting at the origin, and therefore star convex. Thus, a non-star convex snowflake must contain at least one face, so $K = k_{\max}(\mathrm{T}(d, 1), A)$ is a suitable upper bound (for any $d \in \mathcal{D}$). Since $\mathrm{T}(d, 1)$ is star convex, K can be computed in $\mathcal{O}(\log^2 K)$ rounds. And since a triangle fills an area, we have $|V(k \cdot \mathrm{T}(d, 1))| = \Theta(k^2)$, so we have an upper bound of $K = \mathcal{O}(\sqrt{n})$. Together, we obtain the following theorem (see Appendix C for the proof):

▶ **Theorem 23.** *Let A be an amoebot structure and S a snowflake shape. Given a tree representation of S, the amoebots can compute $k = k_{\max}(S, A)$ in a binary counter and determine $\mathcal{V}(k \cdot S^{(r)}, A)$ for all $r \in \{0, \dots, 5\}$ within $\mathcal{O}(\log^2 k)$ rounds if S is star convex and $\mathcal{O}(K \log K)$ rounds otherwise, where $k \leq K = k_{\max}(\mathrm{T}(E, 1), A) = \mathcal{O}(\sqrt{n})$.*

7 Conclusion and Future Work

In this paper, we introduced the shape containment problem for the amoebot model of programmable matter and presented sublinear solutions using reconfigurable circuits. We showed that for some shapes, there is a lower bound of $\Omega(\sqrt{n})$ rounds due to the arbitrary distribution of valid and invalid placements, even if the maximum scale is known. Using efficient methods of transferring information using circuits, we constructed the class of *snowflake* shapes that can be solved in sublinear time and contains a large variety of shapes. For the subset of shapes that are *star convex*, we showed how to solve the problem in polylogarithmic time and proved that binary search is not generally applicable to other shapes.

It is unclear how non-snowflake shapes can be characterized and what exactly distinguishes shapes that are affected by the lower bound from shapes with more efficient solutions, such as star convex shapes. It would be interesting to explore whether the lower bound can be improved when including the scale factor search. Naturally, efficient solutions for arbitrary shapes or other shape classes are of interest as well.

The related problems of finding the smallest scale at which a given shape contains the amoebot structure, as well as finding scales and placements with maximal overlap and minimal difference, could also be investigated to support shape formation algorithms. To expand the set of possible applications further, one could examine other, non-uniform scaling behaviors, perhaps allowing the shapes to maintain fine details at larger scales, similar to fractal shapes.

References

1 Pankaj K. Agarwal, Nina Amenta, and Micha Sharir. Largest Placement of One Convex Polygon Inside Another. *Discrete & Computational Geometry*, 19(1):95–104, 1998. doi:10.1007/PL00009337.

2 Md Abdul Aziz Al Aman, Raina Paul, Apurba Sarkar, and Arindam Biswas. Largest Area Parallelogram Inside a Digital Object in a Triangular Grid. In Reneta P. Barneva, Valentin E. Brimkov, and Giorgio Nordo, editors, *Combinatorial Image Analysis - 21st International Workshop (IWCIA)*, volume 13348 of *LNCS*, pages 122–135, Cham, 2022. Springer. doi:10.1007/978-3-031-23612-9_8.

3 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán, and Stefanie Wuhrer. Reconfiguration of Cube-Style Modular Robots Using O(log n) Parallel Moves. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-92182-0_32.

4 Matthias Artmann, Andreas Padalkin, and Christian Scheideler. On the shape containment problem within the amoebot model with reconfigurable circuits, 2025. doi:10.48550/arXiv.2501.16892.

5 Ahmed Amine Chafik, Jaafar Gaber, Souad Tayane, Mohamed Ennaji, Julien Bourgeois, and Tarek El Ghazawi. From Conventional to Programmable Matter Systems: A Review of Design, Materials, and Technologies. *ACM Comput. Surv.*, 56(8):210:1–210:26, 2024. doi:10.1145/3653671.

6 Bernard Chazelle. The Polygon Containment Problem. *Advances in Computing Research*, 1(1):1–33, 1983.

7 Joshua J. Daymude, Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Christian Scheideler, and Andréa W. Richa. Convex Hull Formation for Programmable Matter. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *21st International Conference on Distributed Computing and Networking*, ICDCN '20, pages 1–10, New York, NY, USA, 2020. ACM. doi:10.1145/3369740.3372916.

8 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: Algorithms and concurrency control. *Distributed Computing*, 36(2):159–192, 2023. doi:10.1007/s00446-023-00443-3.

9 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief Announcement: Amoebot - A New Model for Programmable Matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 220–222, Prague, Czech Republic, 2014. ACM. doi:10.1145/2612669.2612712.

10 Giuseppe A. Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33(1):69–101, 2020. doi:10.1007/s00446-019-00350-6.

11 Yuval Emek, Yuval Gil, and Noga Harlev. On the Power of Graphical Reconfigurable Circuits. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.22.

12 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating Amoebots via Reconfigurable Circuits. *Journal of Computational Biology*, 29(4):317–343, 2022. doi:10.1089/cmb.2021.0363.

13 Melvin Gauci, Radhika Nagpal, and Michael Rubenstein. Programmable Self-disassembly for Shape Formation in Large-Scale Robot Collectives. In Roderich Groß, Andreas Kolling, Spring Berman, Emilio Frazzoli, Alcherio Martinoli, Fumitoshi Matsuno, and Melvin Gauci, editors, *Distributed Autonomous Robotic Systems: The 13th International Symposium*, volume 6 of *Springer Proceedings in Advanced Robotics*, pages 573–586, Cham, 2018. Springer. doi:10.1007/978-3-319-73008-0_40.

14 Kyle W. Gilpin. *Shape Formation by Self-Disassembly in Programmable Matter Systems*. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2012.

15 G. Hansen, I. Herburt, H. Martini, and M. Moszyńska. Starshaped sets. *Aequationes mathematicae*, 94(6):1001–1092, 2020. doi:10.1007/s00010-020-00720-7.

16 Marvin Künnemann and André Nusser. Polygon Placement Revisited: (Degree of Freedom + 1)-SUM Hardness and an Improvement via Offline Dynamic Rectangle Union. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 3181–3201, Alexandria, VA, USA, 2022. SIAM. doi:10.1137/1.9781611977073.124.

17 Thiago de Castro Martins and Marcos de Sales Guerra Tsuzuki. Simulated annealing applied to the irregular rotational placement of shapes over containers with fixed dimensions. *Expert Systems with Applications*, 37(3):1955–1972, 2010. doi:10.1016/j.eswa.2009.06.081.

18 P. McMullen. Sets homothetic to intersections of their translates. *Mathematika*, 25(2):264–269, 1978. doi:10.1112/S0025579300009505.

19 Andreas Padalkin, Manish Kumar, and Christian Scheideler. Reconfiguration and Locomotion with Joint Movements in the Amoebot Model. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2024.18.

20 Andreas Padalkin and Christian Scheideler. Polylogarithmic Time Algorithms for Shortest Path Forests in Programmable Matter. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 65–75, New York, NY, USA, 2024. ACM. doi:10.1145/3662158.3662776.

21 Andreas Padalkin, Christian Scheideler, and Daniel Warner. The Structural Power of Reconfigurable Circuits in the Amoebot Model. In Thomas E. Ouldridge and Shelley F. J. Wickham, editors, *28th International Conference on DNA Computing and Molecular Programming (DNA 28)*, volume 238 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DNA.28.8.

22 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

23 Micha Sharir and Sivan Toledo. Extremal polygon containment problems. *Computational Geometry*, 4(2):99–118, 1994. doi:10.1016/0925-7721(94)90011-6.

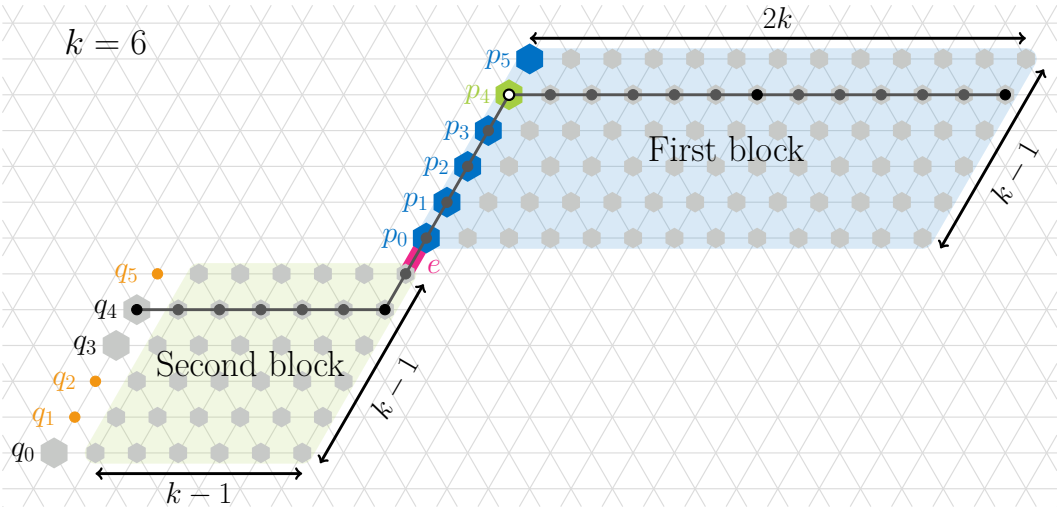


Figure 6 Overview of the amoebot system construction for scale $k = 6$. The first block is shaded blue and the second block is shaded green. The nodes q_i that are not contained in the structure are colored orange. Amoebot p_4 is a valid placement of $k \cdot S$ because q_4 is part of the structure.

24 Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, 2019. doi:10.1016/j.robot.2019.07.012.

25 Tommaso Toffoli and Norman Margolus. Programmable Matter: Concepts and Realization. *International Journal of High Speed Computing*, 05(02):155–170, 1993. doi:10.1142/S0129053393000086.

26 Lidong Yang, Jiangfan Yu, Shihao Yang, Ben Wang, Bradley J. Nelson, and Li Zhang. A Survey on Swarm Microrobotics. *IEEE Transactions on Robotics*, 38(3):1531–1551, 2022. doi:10.1109/TRO.2021.3111788.

A A Lower Bound for Finding Valid Placements

▶ **Theorem 5.** *There exists a shape S_Ω such that for any choice of origin and every amoebot algorithm $\mathcal{A}$ that terminates after $o(\sqrt{n})$ rounds, there exists an amoebot structure A for which the algorithm does not compute $\mathcal{V}(k_{\max}(S_\Omega, A) \cdot S_\Omega, A)$, even if $k_{\max}$ is known.*

Proof. We use the shape S_Ω with a long arm and a short arm connected by a diagonal edge, as depicted in Fig. 4. Let $\mathcal{A}$ be an amoebot algorithm that terminates in $o(\sqrt{n})$ rounds. For every $k \in \mathbb{N}$, we will construct a set A_k of amoebot structures such that $k_{\max}(S_\Omega, A) = k$ for all $A \in A_k$ and only one rotation matches at this scale. Let $k \in \mathbb{N}$ be arbitrary, then we construct A_k as follows (see Fig. 6 for reference):

First, we place a parallelogram of width $2k$ and height $k-1$ with its lower left corner at the origin and call this the *first block*. The first block contains $(2k+1)k = 2k^2 + k$ amoebots and is shared by all $A \in A_k$. Let $p_0, \ldots, p_{k-1}$ be the nodes occupied by the left side of the parallelogram, ordered from bottom to top. Next, we place a second parallelogram with width and height $k-1$ such that its right side extends the first block's left side below the origin. This second block contains k^2 amoebots and is also the same for all structures. It is only connected to the first block by a single edge, e. Let $q_0, \ldots, q_{k-1}$ be the nodes one step to the left of the second block, again ordered from bottom to top.

We define A_k as the set of amoebot structures that consist of these two blocks and m additional amoebots on the positions $q_0, \dots, q_{k-1}$, where $1 \leq m \leq k$. Thus, A_k contains $2^k - 1$ distinct structures. Now, consider placements of S_Ω with maximum scale in any structure $A \in A_k$. For $m = k$, there are exactly k valid placements at scale k, represented by the amoebots $p_0, \dots, p_{k-1}$. The longest continuous lines of amoebots in A have length $2k$ and form the first block. In every valid placement, the longer arm of $k \cdot S_\Omega$ must occupy one of these lines, so no larger scales or other rotations are possible. If q_i is not occupied for some $0 \leq i \leq k-1$, then p_i is not a valid placement because the end of the shorter arm of $k \cdot S_\Omega$ would be placed on q_i. At least one q_i is always occupied, so the maximum scale of S_Ω is k for every $A \in A_k$. Observe that every structure $A \in A_k$ has a unique configuration of valid placements of $k \cdot S_\Omega$: $p \in \mathcal{V}(k \cdot S_\Omega, A)$ if and only if $p = p_i$ and $q_i \in A$ for some $0 \leq i \leq k-1$.

Next, consider the size of the structures in A_k. The maximum number of amoebots is $2k^2 + k + k^2 + k = 3k^2 + 2k$, obtained for $m = k$. This means we have $n \leq 3k^2 + 2k \leq 4k^2$ for large enough k, i.e., $k \geq \sqrt{n}/2$ for all $k \geq 2$ and all $A \in A_k$.

Let $A \in A_k$ be arbitrary and consider the final states of $p_0, \dots, p_{k-1}$ after $\mathcal{A}$ has been executed on A. Each amoebot must be categorized as either a valid or an invalid placement of $k \cdot S_\Omega$. We can assume that this categorization is independent of any randomized decisions because otherwise, there would be a non-zero probability of false categorizations. Thus, the final state depends only on the structure A itself. Recall that structures in A_k only differ in the positions $q_0, \dots, q_{k-1}$ and every path between q_i (or an occupied neighbor) and p_i must traverse the single edge e connecting the two blocks. We can assume that all communication happens via circuits (see Sec. 1.2). Since the first block is the same in all structures, the final states of $p_0, \dots, p_{k-1}$ only depend on the sequence of signals sent from the second block to the first block through e. To compute the correct set of valid placements, each amoebot structure in A_k therefore has to produce a unique sequence of signals: If for any two configurations, the same sequence of signals is sent through e, the final states of $p_0, \dots, p_{k-1}$ will be identical, so at least one will be categorized incorrectly.

Let c_ϕ be the number of pins used by $\mathcal{A}$. Then, the number of different signals that can be sent via one edge in one round is $2^{c_\phi} = \mathcal{O}(1)$, and the number of signal sequences that can be sent in r rounds is 2^{rc_ϕ}. Therefore, to produce at least $2^k - 1$ different sequences of signals, we require $r = \Omega(k/c_\phi) = \Omega(\sqrt{n})$ rounds. By the assumption that $\mathcal{A}$ terminates after $o(\sqrt{n})$ rounds, $\mathcal{A}$ will produce at least one false result for sufficiently large k.

It remains to be shown that the same arguments hold for all equivalent versions of S_Ω that contain the origin. If the origin is placed on another node of the longer arm, the valid placement candidates $p_0, \dots, p_{k-1}$ are shifted to the right by k or $2k$ steps, respectively; everything else remains the same. If the origin is placed on the shorter arm of the shape, we switch the roles of the first and the second block. We place amoebots on all positions $q_0, \dots, q_{k-1}$ and use the right side of the first block as the controlling positions instead. The number and size of the resulting amoebot structures remain the same, so the same arguments hold as before. ◀

B Self-Contained Shapes are Star Convex

To show the properties of star convex shapes, we will use the following equivalent characterization:

▶ **Lemma 24.** *A shape S is star convex with its origin as a center node if and only if S is the union of parallelograms of the form $\mathrm{L}(d, \ell) \oplus \mathrm{L}(d', \ell')$ and convex shapes of the form $\mathrm{T}(d, 1) \oplus \mathrm{L}(d, \ell) \oplus \mathrm{L}(d', \ell')$, where d' is obtained from d by a $60°$ clockwise rotation. The number of these shapes is linear in $|V(S)|$.*

For the proof of Theorem 21, we first show several lemmas that provide the necessary tools. To start with, we show that non-star convex shapes cannot become star convex by scaling, and for a sufficiently large scale, every center candidate has a shortest path with a missing edge. It is clear that star convex shapes stay star convex after scaling (by Lemma 24), but non-star convex shapes gain new potential center nodes, making it less obvious why there still cannot be a center.

▶ **Lemma 25.** *Let S be an arbitrary non-star convex shape, then for every $k \in \mathbb{N}$, $k \cdot S$ is not star convex, and for $k > 2$ and every node $c \in V(k \cdot S)$, there exists a shortest path from c to a node $v \in V(k \cdot S)$ in G_Δ with at least one edge not contained in $k \cdot S$.*

Next, we show a *fixed point* property that is particularly useful for scales k and $k+1$.

▶ **Lemma 26.** *Let S be a shape and $k \in \mathbb{N}$ a scale such that there exists a $t \in V_\Delta$ with $k \cdot S + t \subseteq (k+1) \cdot S$. Then, t must be in $V(S)$ and we call t a* fixed node *of S. Further, for every node $v \in V(S)$, a shortest path from $k \cdot v + t$ to $(k+1) \cdot v$ is also a shortest path from t to v and vice versa.*

Finally, we eliminate the need for covering rotations by showing that for sufficiently large scales k and $k+1$, $k \cdot S^{(r)}$ does not fit into $(k+1) \cdot S$ for any $r \in \{1, \dots, 5\}$ unless S is rotationally symmetric.

▶ **Definition 27.** *A shape S is called* rotationally symmetric with respect to $r \in \{1, 2, 3\}$ *(or* r-symmetric*) if there exists a translation $t \in V_\Delta$ such that $S^{(r)} + t = S$.*

Note that $r \in \{1, 2, 3\}$ covers all possible rotational symmetries in the triangular grid and 1-symmetry is equivalent to 2- and 3-symmetry combined. Additionally, the translation t is unique. Also note that 1-symmetry is more commonly called 6-*fold* symmetry, 2-symmetry is known as 3-fold symmetry and 3-symmetry is known as 2-fold symmetry.

▶ **Lemma 28.** *Let S be a shape and $r \in \{1, 2, 3\}$ be arbitrary. If S is not r-symmetric, then there is a scale $k_0 \in \mathbb{N}$ such that for every $k \geq k_0$, $k \cdot S^{(r)}$ does not fit into $(k+1) \cdot S$.*

Using these lemmas, we can now prove the main theorem about self-contained and star convex shapes:

▶ **Theorem 21.** *A shape is self-contained if and only if it is star convex.*

Proof. First, let S be star convex with center node c and consider two scale factors $k < k'$. By Lemma 24, S can be represented as

$$S = \left(\bigcup_{i=1}^{m_1} P_i \cup \bigcup_{j=1}^{m_2} T_j \right) + c,$$

where the P_i are parallelograms and the T_i are Minkowski sums of parallelograms with triangles. Then, we have $k \cdot P_i \subseteq k' \cdot P_i$ and $k \cdot T_j \subseteq k' \cdot T_j$ since each of the P_i and T_j is a convex shape. We choose t such that $k \cdot c + t = k' \cdot c$, then each of the constituent shapes of $k \cdot S + t$ is contained in its counterpart in $k' \cdot S$. This already shows that every star convex shape is self-contained.

Now, let S be a shape that is not star convex. We will show that S is not self-contained by finding a scale $k \in \mathbb{N}$ such that for every $t \in \mathbb{R}^2$ and $r \in \{0, \dots, 5\}$, $(k \cdot S^{(r)} + t) \setminus ((k+1) \cdot S) \neq \emptyset$. Using Lemma 25, we can assume that for every node $c \in V(S)$, there exists a shortest path Π to a node $v \in V(S)$ such that at least one edge of Π is not contained in S. If this is not the case for S already, we simply consider $3 \cdot S$ as the new base shape.

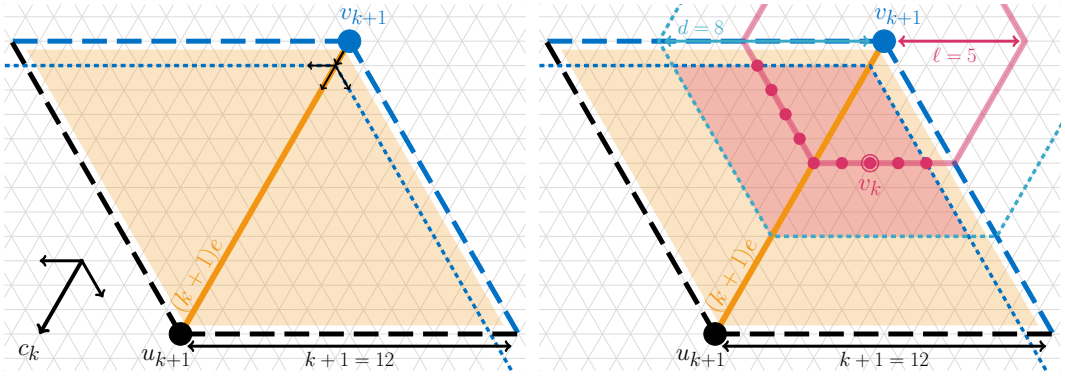


Figure 7 Illustration of the constraints forcing v_k to lie in a region unoccupied by $(k+1)\cdot S$ for $k=11$. The scaled unoccupied edge $(k+1)\cdot e$ and the empty parallelogram region it produces are highlighted and shaded in orange. The parallelogram's edges are drawn as dashed lines of length $k+1$, incident to u_{k+1} and v_{k+1}. At most these edges of the parallelogram can be occupied by $(k+1)\cdot S$. The distance between c and v resp. v_k and v_{k+1} is $\ell=5<k+1$. In the bottom left corner of (a), the directions used by all shortest paths from v_{k+1} to v_k (resp. v to c) are shown, with the opposite direction of e emphasized because at least one such edge must be on every path. This prevents v_k from lying on an edge incident to v_{k+1}, shown by the dotted blue lines and the small black arrows. Similarly, v_k cannot lie on any of the dashed black edges incident to u_{k+1} because it must be closer to v_{k+1}. In particular, its distance to v_{k+1} is bounded by $d=8$, as indicated by the light blue dotted lines in (b). The red lines show the nodes with distance exactly ℓ to v_{k+1}. By combining all constraints, v_k has to be one of the red nodes in the dark shaded area, none of which are occupied by $(k+1)\cdot S$.

By Lemma 28, we can find k_0 large enough that no non-zero rotation of $k\cdot S$ fits into $(k+1)\cdot S$ for any scale $k\geq k_0$ unless S is rotationally symmetric. In this case, however, the rotated version of S is the same as a translation of S, so we can disregard rotations altogether because they do not affect the existence of valid translations of $k\cdot S$ in $(k+1)\cdot S$.

Let H be the convex hull of S and let $d\in\mathbb{N}$ be its diameter. Consider any scale $k>d$ and some placement $c\in V_\Delta$ such that $k\cdot S+c\subseteq(k+1)\cdot S$. By Lemma 26, $c\in V(S)$. As argued above, there is a node $v\in V(S)$ such that a shortest path Π from c to v has at least one edge that is not contained in S. We choose v and Π such that the last edge is missing, w.l.o.g. Let $u\in V(H)$ be the predecessor of v on Π, i.e., the missing edge e of Π connects u and v. Since S does not contain e, it cannot contain the two incident faces. Thus, there is an area in the shape of a regular parallelogram with side length $k+1$ whose diagonal is $(k+1)\cdot e$ and whose interior does not intersect $(k+1)\cdot S$, i.e., only its edges could be covered by edges of $(k+1)\cdot S$ (see Fig. 7). These edges are incident to $v_{k+1}=(k+1)\cdot v$ and $u_{k+1}=(k+1)\cdot u$ and they lie on different axes than e. By Lemma 26, the grid distance between $v_k=k\cdot v+c$ and v_{k+1} is the length of Π, which is bounded by d. By the choice of $k>d$, the distance between u_{k+1} and v_{k+1} (which is $k+1$) is therefore greater than the distance between v_k and v_{k+1}. Now, observe that for any node on one of the parallelogram's edges incident to u_{k+1}, the distance to v_{k+1} remains $k+1$. Thus, v_k cannot lie on any of these two edges.

Furthermore, recall from Lemma 26 that any shortest path from c to v is also a (translated) shortest path from v_k to v_{k+1}. Because such a path contains at least one edge parallel to e, every shortest path from v_{k+1} to v_k contains at least one edge in the opposite direction

of e. Therefore, since the edges of the empty parallelogram that are incident to v_{k+1} lie on different axes than e, v_k cannot lie on these two edges either. The direction of e also implies that v_k must be on the side of the two edges that is closer to u_{k+1}.

Together, these constraints imply that v_k lies *inside* the parallelogram that is not contained in $(k+1)\cdot S$, so the placement identified by c does not satisfy $k\cdot S+c \subseteq (k+1)\cdot S$, contradicting our assumption. Since this works for every choice of $c \in V(S)$, there is no placement of $k \cdot S$ in $(k+1) \cdot S$. ◀

C Main Theorem

▶ **Theorem 23.** *Let A be an amoebot structure and S a snowflake shape. Given a tree representation of S, the amoebots can compute $k = k_{\max}(S, A)$ in a binary counter and determine $\mathcal{V}(k \cdot S^{(r)}, A)$ for all $r \in \{0, \dots, 5\}$ within $\mathcal{O}(\log^2 k)$ rounds if S is star convex and $\mathcal{O}(K \log K)$ rounds otherwise, where $k \leq K = k_{\max}(\mathrm{T}(E, 1), A) = \mathcal{O}(\sqrt{n})$.*

Proof. The amoebots first establish binary counters on all maximal segments in A, for all axes. At least one of these will be large enough to store $k = k_{\max}(S, A)$ as long as S is non-trivial. In the following, they use all these counters simultaneously and deactivate every counter exceeding its memory during an operation.

Consider the case where S is star convex. By Lemma 8, combined with Lemma 19, the amoebots can compute $k_{\max}$ using a binary search and find all valid placements at all six rotations within $\mathcal{O}(\log^2 k)$ rounds. Now, let S be a snowflake that is not star convex. In this case, S must contain at least one triangular face because all snowflakes without faces are unions of lines meeting at the origin, which are star convex (observe that the Minkowski sum of an edge with a line on a different axis always contains some faces). Then, $K = k_{\max}(\mathrm{T}(\mathrm{E}, 1), A)$ is an upper bound for $k_{\max}(S, A)$. The amoebots can compute K within $\mathcal{O}(\log^2 K)$ rounds and store it in binary counters, since triangles are star convex. A simple linear search for $k_{\max}$ yields the runtime of $\mathcal{O}(K \log K)$ by Lemma 6. $K = \mathcal{O}(\sqrt{n})$ follows from the fact that the number of nodes covered by $k \cdot \mathrm{T}(\mathrm{E}, 1)$ grows quadratically with k. ◀

Auditable Shared Objects: From Registers to Synchronization Primitives

Hagit Attiya
Technion – Israel Institute of Technology, Haifa, Israel

Antonio Fernández Anta
IMDEA Software Institute & IMDEA Networks Institute, Madrid, Spain

Alessia Milani
Aix Marseille Univ, CNRS, LIS, Marseille, France

Alexandre Rapetti
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Corentin Travers
Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

Auditability allows to track operations performed on a shared object, recording who accessed which information. This gives data owners more control on their data. Initially studied in the context of single-writer registers, this work extends the notion of auditability to other shared objects, and studies their properties.

We start by moving from single-writer to *multi-writer* registers, and provide an implementation of an *auditable n-writer m-reader read / write register*, with $O(n + m)$ step complexity. This implementation uses $(m + n)$-sliding registers, which have consensus number $m + n$. We show that this consensus number is necessary. The implementation extends naturally to support an *auditable load-linked / store-conditional* (LL/SC) shared object. LL/SC is a primitive that supports efficient implementation of many shared objects. Finally, we relate auditable registers to other access control objects, by implementing an *anti-flickering deny list* from auditable registers.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Auditability, Wait-free implementation, Synchronization power, Distributed objects, Shared memory, LL/SC, Deny List

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.8

Related Version *Full Version*: `https://arxiv.org/abs/2508.14506` [5]

Funding *Hagit Attiya*: Supported by the Israel Science Foundation (22/1425 and 25/1849).
Antonio Fernández Anta: Supported by project PID2022-140560OB-I00 (DRONAC) funded by MICIU/AEI /10.13039/501100011033 and ERDF, EU.
Alessia Milani: Supported in part by ANR project TRUSTINCloudS (ANR-23-PECL-0009).
Corentin Travers: Supported in part by ANR project DUCAT (ANR-20-CE48-0006).

1 Introduction

The ability to track operations on shared objects is a fundamental feature in distributed systems. *Auditability*, in particular, enables data owners to monitor access to their data by recording who accessed which information. This mechanism provides a robust alternative to traditional access control mechanisms, shifting the emphasis from access restriction to post-hoc accountability. Auditability is helpful for preserving data privacy, as it can be used after a data breach, to enforce accountability for data access. This is particularly useful in shared, remotely accessed storage systems, where, for instance, understanding the extent of a data breach can help mitigate its impact.


© Hagit Attiya, Antonio Fernández Anta, Alessia Milani, Alexandre Rapetti, and Corentin Travers;
licensed under Creative Commons License CC-BY 4.0



Auditability has been initially studied in the context of non-atomic replicated storage [8] and single-writer atomic registers [6]. This work explores auditability for *other shared objects*, considering the following key questions: (1) What types of base objects are necessary and sufficient to implement auditable *multi-writer* atomic registers? (2) How does auditability impact the design of *common synchronization primitives*? (3) Can auditable registers be leveraged to construct more complex *access control mechanisms*?

We begin to answer these questions by presenting a wait-free, linearizable implementation of an auditable n-writer m-reader read/write register (abbreviated as (m, n)-auditable register), with linear (in $n + m$) step complexity. To achieve these properties, our algorithm uses sliding registers [19]: a *k-sliding register* keeps an ordered list of the latest k values written to it. If each reader and writer writes at most once in an $(m + n)$-sliding register, the sliding register can be used to uniquely order the operations. This implies a linearization of the read and write operations on the auditable register. The key challenge is, therefore, how to deal with multiple read and write operations on the auditable register. The backbone of our (m, n)-auditable register implementation is a sequence of $(m + n)$-sliding registers, each associated with a different value written to the auditable register. The sliding registers are used to agree which write operation will set the next value of the register, and to track which readers have read this value. An additional challenge is to be able to efficiently find the "current" sliding register when reading and writing, to maintain bounded step complexity.

The consensus number of $(m + n)$-sliding registers is exactly $m + n$. (I.e., they allow to solve consensus among exactly $m + n$ processes.) Thus, consensus number $m + n$ is sufficient for implementing an (m, n)-auditable register. We prove that it is also necessary, by showing that an (m, n)-auditable register can be used to solve consensus among $m + n$ processes.

Beyond registers, we explore auditability for more expressive synchronization primitives, like *load-linked/store-conditional* (LL/SC) objects [17]. The LL operation reads a value from a memory location, while the SC operation writes a new value to the same location only if no process has written to it since the LL operation. If another process has changed the value, the SC fails, requiring the operation to be retried. LL/SC enables the efficient construction of a wide range of non-blocking data structures [10]. We leverage the algorithmic ideas of our auditable register to construct an algorithm that implements an LL/SC object for n processes, using $2n$-sliding register.

Finally, we relate auditable registers to Deny Lists [11], showcasing how auditability can be leveraged to build security primitives. *Allow Lists* and *Deny Lists*, as defined by Frey, Gestin, and Raynal [11], are two access control objects that give designated processes (called *managers*) the ability to grant and/or revoke access rights for a given set of resources. Their work specifies and investigates the synchronization power of Allow Lists and Deny Lists; the latter in two flavors, with and without an *anti-flickering* property. (An anti-flickering Deny List ensures that transient revocations do not undermine long-term access control policies.)

We give a relatively simple specification of an *immediate* Deny List, which is stronger than the anti-flickering Deny List of Frey et al., and show that it can be efficiently implemented from auditable registers. The step complexity of the resulting implementation (polynomially) depends on the number of resources and the number of processes. In contrast, while being wait-free, the step complexity of the anti-flickering Deny List implementation in [11] grows with the number of operations, and is, essentially, unbounded.

Related Work

Auditability was introduced by Cogo and Bessani [8] in the context of replicated registers. They considered a register as an abstraction for distributed storage that provides read and write operations to clients. Cogo and Bessani define auditability in terms of two properties:

completeness ensures that all readers' data access are detected, while *accuracy* ensures that readers who do not access data are not wrongly incriminated. They present an algorithm to implement an auditable *regular* (non-atomic) register, using $n \geq 4f + 1$ atomic read/write shared objects, f of which may be faulty (writers and auditors fail only by crashing; faulty readers may be Byzantine). Their implementation relies on information dispersal schemes, where the input of a high-level write is split into several pieces, each written in a different low-level shared object. Each low-level shared object keeps a trace of each access, and in order to read, a process has to collect sufficiently many pieces of information in many low-level shared objects, which allows to audit the read.

In asynchronous message-passing systems where f processes can be Byzantine, Del Pozzo, Milani, and Rapetti [9] study the possibility of implementing an atomic auditable register. They prove that without communication between servers, auditability requires at least $4f + 1$ servers. They also show that allowing servers to communicate with each other admits an auditable atomic register with optimal resilience of $3f + 1$.

The auditability definition of [8] is tightly coupled with their multi-writer, multi-reader register emulation in a replicated storage system using an information-dispersal scheme. An implementation-agnostic auditability definition was later proposed by Attiya et al. [6], based on collectively linearizing read, write, and audit operations. They show that auditing adds power to reading and writing, as it allows to solve consensus, implying that auditing requires strong synchronization primitives. They also give several implementations that use non-universal primitives (like *swap* and *fetch&add*), for a single writer and either several readers or several auditors (but not both).

Recent work [4] have extended auditability to ensure that even a curious (but honest) reader cannot effectively read data without being audited; in fact, curious readers cannot even audit other readers. The paper provides implementations of several auditable objects, including registers. However, the constructions depend on universal primitives, including *compare&swap*, in contrast to our constructions, which only employ sliding registers, whose consensus number is bounded.

Alhajaili and Jhumka [1] study an interesting variant of auditability with malicious processes, with and without a trusted party. Their definition of auditability has to do with the ability of determining if a system runs as specified, and determine the incorrect behavior in case it does not. This involves systematically tracking and recording system activities to facilitate fault detection and diagnosis. The authors propose methodologies for integrating auditability into system design, emphasizing its role in simplifying the debugging process and improving overall system robustness.

Our algorithms (like most prior work) maintain a large amount of auditing information, which grows with the number of operations performed in an execution. Hajisheykhi, Roohitavaf, and Kulkarni [13] investigate how to reduce the space used for saving auditing data (to be used to restore the state after a fault), by leveraging causal dependencies among them. They propose protocols for ensuring accountability in distributed systems when auditable events – deviations from standard protocols – occur. They introduce two self-stabilizing protocols: an unbounded state space protocol that propagates auditable events across all nodes and a more efficient bounded state space protocol that achieves the same awareness without increasing resource demands. Their approach ensures that before a system can recover from an auditable event, all processes are aware of it, preventing unauthorized restorations and reinforcing accountability.

Access control objects regulate access to resources by defining policies for granting or denying permissions. Some of the most common access control mechanisms include Access Control Lists [21], which specify allowed or denied actions for individual users or processes,

and Capability-Based Access Control [18], where access is granted through transferable tokens rather than predefined rules. Role-Based Access Control [20] simplifies management by assigning permissions to roles instead of individuals, making it widely used in enterprise environments, while Attribute-Based Access Control [15] dynamically evaluate attributes such as user location, device type, and time of access. AllowLists and DenyLists [11] further control access by explicitly specifying permitted or blocked users or processes, often used in security filters and authentication systems.

An area related to auditability is *data provenance* (also called *lineage*) in databases [7, 12], which involves tracking the origin and transformations of data, focusing on its lineage across different processes or systems. Provenance is concerned with the integrity, quality, and reproducibility of data as it moves through various stages, such as aggregation or computation. While both auditability and provenance track interactions with data, auditability is more focused on recording who performed an operation, primarily for access control and security, while provenance aims to provide transparency about the history and transformations of the data for purposes such as verification and accountability in data analysis.

Summary of Our Contributions and Organization of the Paper

- We extend auditability from single-writer to multi-writer registers and characterize the necessary and sufficient consensus number needed for their implementation.
- We introduce and implement auditable LL/SC objects, demonstrating auditability can be provided beyond read / write operations.
- We construct an anti-flickering Deny List object from auditable registers, illustrating the connection between auditability and access control mechanisms.

Section 2 presents our model and formally defines auditable shared objects. Sections 3 and 4 prove the necessary and sufficient conditions for auditable multi-writer registers, respectively. Section 5 describes how to implement auditable LL/SC objects. Section 6 presents the algorithm to implement an anti-flickering Deny List object with auditable registers. Finally, Section 7 concludes with open questions and future directions. Proofs omitted due to space constraints can be found in the full version [5].

2 Definitions

We use a standard model, in which a set of processes $p_1, \ldots, p_n$, communicate through a shared memory consisting of *base objects*. The base objects are accessed with *primitive operations*. In addition to atomic registers, our implementations use k-sliding registers, whose consensus number is exactly k [19]. Specifically, a *k-sliding register* [19] stores the sequence of the last k values written to it (or the last x values when only $x < k$ values have been written). A *write* with input v appends v at the end of the sequence and removes the first one if the sequence is already of size k. A *read* operation returns the current sequence. A standard read / write register is a 1-sliding register.

An *implementation* of a (high-level) object T specifies a program for each process and each operation of the object T; when receiving an *invocation* of an operation, the process takes *steps* according to this program. Each step by a process consists of some local computation, followed by a single primitive operation on a base object. The process may change its local state after a step, and it may return a *response* to the operation of the high-level object.

In order not to confuse operations performed on the implementation of the high-level object T and primitives applied to base objects, the former are denoted with capital letters and the later in normal font.

A *configuration* C specifies the state of every process and of every base object. An *execution* α is an alternating sequence of configurations and events, starting with an *initial configuration*; it can be finite or infinite. An operation *completes* in an execution α if α includes both the invocation and response of the operation; if α includes the invocation of an operation, but no matching response, then the operation is *pending*. An operation *op precedes* another operation op' in α if the response of *op* appears before the invocation of op'.

A *history* H is a sequence of invocation and response events. The notions of *complete*, *pending* and *preceding* operations extend naturally to histories.

The standard correctness condition for concurrent implementations is *linearizability* [14]: intuitively, it requires that each operation appears to take place instantaneously at some point between its invocation and its response. Formally:

▶ **Definition 1.** *Let $\mathcal{A}$ be an implementation of an object T. An execution α of $\mathcal{A}$ is* linearizable *if there is a sequential execution $\lambda(\alpha)$ (a* linearization *of the operations on T in α) such that:*

- *$\lambda(\alpha)$ contains all complete operations in α, and a (possibly empty) subset of the pending operations in α (completed with response events),*
- *If an operation op precedes an operation op' in α, then op appears before op' in $\lambda(\alpha)$, and*
- *$\lambda(\alpha)$ respects the sequential specification of the high-level object T.*

$\mathcal{A}$ is linearizable *if all its executions are linearizable.*

An implementation is *wait-free* if, whenever there is a pending operation by process p, this operation returns in a finite number of steps by p.

An *auditable register* supports, in addition to the standard READ and WRITE operations, also an AUDIT operation that reports which values were read by each process [6]. An AUDIT has no parameters and it returns a set of pairs, (j, v), where j is a process id, and v is a value of the register. A pair (j, v) indicates that process p_j has read the value v. The sequential specification of an auditable register enforces, in addition to the usual specification of READ and WRITE operations, that a pair appears in the set returned by an AUDIT operation if and only if it corresponds to a preceding READ operation.

We implement a *load-linked / store-conditional* (LL/SC) variable, supporting the following operations: LL(x) returns the value stored in x, and SC(x, new) writes the value *new* to x, if it was not written since the last LL(x) performed by the process; otherwise, x is not modified. SC returns *true* if it writes successfully, and *false* otherwise. An *auditable LL/SC variable adds an* AUDIT *operation, whose sequential specification* returns a set of process-value pairs, corresponding to preceding LL operations.

3 Consensus number of n-Writer, m-Reader Auditable Register $\geq m + n$

An (m, n)-auditable register can be written by n processes, read by m processes, and be audited by all processes. An audit operation returns a set of pairs (p, v) where p is a process id, and v is a value. This set holds the values returned by the read operations that precede the audit.

▶ **Theorem 2.** *For every pair of integers $m, n > 0$, there is an $(m + n)$-process consensus algorithm using (m, n)-auditable registers.*

Proof. The proof is by induction on $\ell = n + m$. The base case, $\ell = 2$, is proved in [6, Proposition 19].

For the induction step, assume the lemma holds for all pairs of values n', m', such that $n' + m' = \ell \geq 2$, and we prove it for $\ell + 1 > 2$. Pick $m > 0, n > 0$ such that $n + m = \ell + 1$; note that $m, n \leq \ell$. To solve consensus among $\ell + 1$ processes $p_1, \ldots, p_{\ell+1}$, we partition the processes into two sets: R (the *readers*) of size m, and W (the *writers*) of size n. Since $m > 0$ and $n > 0$, both sets R and W are non-empty.

Each process $p_i, 1 \leq i \leq \ell + 1$ has a standard single-writer multi-reader register S_i. By the induction hypothesis, since $n \leq \ell$, the n writers can agree on one of their proposals with an n-process consensus that uses $(1, n-1)$-auditable registers. Similarly, the m readers can agree on one of their proposal using $(m-1, 1)$-auditable registers. Each reader and writer process p_i writes the value agreed upon in its register S_i.

We use now one (m, n)-auditable register, AR, whose initial value is $\perp$. Writers and readers access AR to select one of the two consensus values. Each writer $\in W$ writes $\top$ to AR and each reader $\in R$ performs a read operation on AR. Writers and readers then audit AR. If the set returned is empty or contains no pair $(p_r, \perp)$ where $r \in R$, then the first operation performed on AR is a write and the writers win. Then, the decision for consensus can be read from at least one of the registers of the writers. Otherwise, the audit operation returns a set containing a pair $(p_r, \perp)$ with $r \in R$. In that case, the first operation performed on AR is a read operation and the readers win. As in the previous case, the decision for consensus can be found by reading the readers' registers. ◀

4 Implementing an n-Writer m-Reader Auditable Register Using $(n + m)$-Sliding Registers

We present a wait-free and linearizable implementation of an n-writer m-reader auditable register; it can support any number of auditors. The implementation uses $(m + n)$-sliding registers. Since the consensus number of (m, n)-auditable registers is $m + n$ (Theorem 2), objects with consensus number $\geq m + n$, like $(m + n)$-sliding registers, are required. (Objects with consensus number ∞, like *compare&swap*, can also be used [4], but our goal is to use objects with the minimal consensus number.) Our main result is:

▶ **Theorem 3.** *There is a wait-free linearizable implementation of an m-reader, n-writer auditable register from $(m + n)$-sliding registers. The step complexity of each operation is in $O(m + n)$.*

4.1 The Algorithm

An implementation of an auditable register has to keep track of the latest value written to the register as well as, for each written value, its set of readers. This can be easily achieved using a single sliding register SLR, provided that its window is unbounded. Such a register hence stores the complete sequence of values written to it, ordered by the oldest first but has infinite consensus number. To perform a WRITE(v), a writer simply writes v to SLR. For auditing purpose, a reader p_i first writes its identifier i to the sliding register, before reading it. The value returned by this READ is then the nearest non-identifier value in the sequence read from SLR previous to the identifier i written by p_i. The reader set of each value can easily be inferred from the sequence stored in SLR. For a value v in the sequence, its reader set is the set of processes whose identifiers follows v, and are before the first non-identifier value that succeeds v, if any.

Our implementation (Algorithms 1 and 2) is based on this simple idea, but instead of a single sliding register with an unbounded window, we use an unbounded array $SLR[-1, \ldots]$ of sliding registers, each with a bounded window of size $m + n$. In order not to confuse

identifiers and written values, WRITE operations insert into the sliding registers *w-tuples* of the form (w, j, v, h), where j the identifier of the writer, v is the input value of the WRITE operation, and h a *helping set* of readers' identifiers (whose role will be explained later).

Each sliding register $SLR[x]$ contains initially the empty sequence (), except the first $SLR[-1]$ whose sequence contains the w-tuple $(\mathsf{w}, j_0, v_0, \emptyset)$ where v_0 is the initial value of the auditable register and j_0 an arbitrary writer's identifier. At any point in the execution, the current value v of the auditable register is found in the sliding register with highest index x whose sequence contains a w-tuple. Specifically, v is the value contained in the *first* w-tuple in the sequence stored in $SLR[x]$. Similarly to the basic implementation sketched above, readers of this value (if any) are the processes whose identifiers appear *before* any w-tuple in the *next* sliding register $SLR[x+1]$.

Therefore, reading or writing the auditable register involves finding the *valid* sliding register (that is, one that does not contain a w-tuple) with lowest index. If this is not done with care, progress of some operation may be lost. For example, the same writer may be always the first to write in each sliding register (i.e., each non empty sequence in any sliding register starts with a w-tuple posted by this writer), thus preventing READ operations from completing or other writers from changing the value of the auditable register. We also have to make sure that each writer or reader writes at most once to each sliding register. Otherwise, the auditing may become inaccurate (as readers identifiers may be removed from some sequence), or the current value of the auditable register may be lost.

To solve theses challenges, the implementation combines the following ideas:

First, we observe that a WRITE operation can terminate after writing in a given sliding register $SLR[x]$, even if its corresponding w-tuple is not the first in the sequence held in $SLR[x]$, provided that that it is concurrent with the WRITE operation that writes first its w-tuple into $SLR[x]$. Indeed, in that case, the WRITE operations that are late to post their w-tuple may be linearized immediately *before* the WRITE whose w-tuple is first. Accordingly, the value, denoted v_x, in the first w-tuple in the sequence stored in $SLR[x]$ is said to be *visible*. The values in the other w-tuple are never returned by any READ operation, and are thus *invisible*.

Second, we use a *max register* M to store the current smallest index $widx$ of the still valid sliding registers. Recall that a max register retains the largest value written to it; wait-free and linearizable max registers can be implemented from atomic read/write registers with linear step complexity [3]. A WRITE thus starts by retrieving this index from M, writes its w-tuple in $SLR[widx]$, and finally updates M with the new index $widx+1$. This ensures that, if several WRITE operations post w-tuples in the same sliding register $SLR[x]$, they are concurrent.

In addition, M stores an m-vector $ridx$ of indexes of sliding registers (which is part of the helping mechanism described next) and, for convenience, the $auditset$ of the values written whose set of readers is already completely determined (the set of readers of each visible value v_x, for $x < widx$). Triples $(widx, ridx, auditset)$ are ordered by their first field. An AUDIT operation therefore reads M, and for $M.widx = x$, gets the *definitive* readers of values $v_{x'}, x' \leq x-2$ from $M.auditset$, to which it adds the possibly *non-definitive* set of readers of v_{x-1} by reading $SLR[x]$.

Finally, a helping mechanism is used to make sure that READ operations are wait-free. The set of readers of the visible value v_x (that appears in the first w-tuple stored in $SLR[x]$) are the processes whose identifiers are before any w-tuple in $SLR[x+1]$ *or* are in the helping set h of the first w-tuple of $SLR[x+1]$. An additional array H indicates, for each reader, the index of the latest sliding register in which it attempted to write its identifier before

Algorithm 1 Multi-writer auditable register: READ and WRITE and AUDIT.

```
1: shared variables
2:     M: a max-register storing a triple (widx, ridx, auditset) ordered by their first field
3:         widx initially 0                                          ▷ index of sliding registers
4:         ridx array of size m, initially [−1, . . . , −1]
                  ▷ ridx[i] highest index of sliding register in which a READ by p_i is recorded
5:         auditset set of pair (process,value), initially ∅
6:     SLR[−1, 0, .. + ∞]: unbounded array of (m + n)-sliding registers,
           initially, SLR[−1] = ((w, j_0, v_0, ∅)), SLR[ℓ] = () for all ℓ ≥ 0
7:     H[1..m]: array of SWMR register, one per reader, initially [−1, . . . , −1]
8: local variables: reader
9:     lsr ← −1; lval ← ⊥                ▷ index of last sliding register read and last value read
10: function READ( )                                  ▷ code for reader process p_i, i ∈ {1, . . . , m}
11:     if lsr ≥ 0 then window ← SLR[lsr].read()          ▷ check for new WRITE since last READ
12:         if there is no w-tuple (w, _, _, _) in window then return lval          ▷ no new WRITE
            (widx, ridx, auditset) ← M.read()      ▷ new WRITE, check if it needs help to complete
13:         if widx = lsr then
14:             for each j ∈ READERS(window) do
15:                 ridx[j] ← lsr; auditset ← auditset ∪ {(j, lval)}
16:             M.writeMax(lsr + 1, ridx, auditset)
17:     repeat
18:         (widx, ridx, auditset) ← M.read()
19:         if ridx[i] > lsr then                                                  ▷ found help
20:             lsr ← ridx[i]; lval ← GETVALUE(lsr − 1) return lval
21:         lsr ← widx; H[i].write(lsr)                ▷ catch up with writers, announce attempt
22:         SLR[lsr].write(i); window ← SLR[lsr].read(); lval ←GETVALUE(lsr − 1)
23:         if ∃(w, _, _, _) ∈ window then          ▷ help corresponding WRITE to complete
24:             for each j ∈ READERS(window) do
25:                 ridx[j] ← lsr; auditset ← auditset ∪ {(j, lval)}
26:             M.writeMax(lsr + 1, ridx, auditset)
27:     until i ∈ READERS(window)
28:     return lval
29: function WRITE(v)                                 ▷ code for writer p_j, j ∈ {1, . . . , n}
30:     widx, ridx, auditset ← M.read(); to_help ← ∅
31:     for all i ∈ {1, . . . , m} do aidx_j ← H[j].read()          ▷ help ongoing read operations
32:         if ridx[j] < aidx_j then                                  ▷ reader p_j may need help
33:             window ← SLR[aidx_j].read()
34:             if j ∉ READERS(window) then to_help ← to_help ∪ {j}
35:     SLR[widx].write(w, j, v, to_help);                                  ▷ announce new write
36:     window ← SLR[widx].read(); val ← GETVALUE(widx − 1)
37:     for each j ∈ READERS(window) do ridx[j] ← widx; auditset ← auditset ∪ {(j, val)}
38:     M.writeMax(widx + 1, ridx, auditset); return
39: function AUDIT( )
40:     widx, _, auditset ← M.read()                  ▷ readers of val with seq. num ≤ widx − 2
41:     window ← SLR[widx].read(); val ← GETVALUE(widx − 1)
42:     auditset ← auditset ∪ {(j, val) : j ∈ READERS(window)}      ▷ readers of val with seq. num
    widx − 1
43:     return auditset
```

Algorithm 2 Multi-writer auditable register: auxiliary functions GETVALUE and READERS.

```
44: function GETVALUE(sn)
45:     window ← SLR[sn].read(); let (w, id, val, _) be the first w-tuple (w, _, _, _) in window
46:     return val
47: function READERS(window)
48:     readers ← {j : there is no w-tuple (w, _, _, _) preceding j in window}
49:     if ∃(w, _, _, _) ∈ window then
50:         let (w, id, _, h) be the 1st (w, _, _, _) in window; readers ← readers ∪ h
51:     return readers
```

any w-tuple. The array $ridx$ (stored, as seen above, in the max register M), records for each reader the highest index of a sliding register in which it has received help (that is, the identifier of the reader is included in the helping set of the first w-tuple of that sliding register). A writer p_j hence determines if a given reader p_i needs help by comparing $H[i]$ and $M.ridx[i]$. If $M.ridx[i] < H[i]$, p_i has an ongoing READ operation, and i is therefore added to the helping set of the w-tuple of the writer. Similarly, reader p_i discovers if it has received help by comparing $M.ridx[i]$ with the index of the latest sliding register in which it writes its identifier.

Hence, besides w-tuples, a given sliding register $SLR[x]$ may also include identifiers i of readers, with the following meaning:

1. If i appears *before* the first w-tuple in $SLR[x]$, then p_i's READ returns v_{x-1}, which is the value stored in the first w-tuple in $SLR[x-1]$.
2. If i appears in the helping set of the first w-tuple in $SLR[x]$, then p_i's READ also returns v_{x-1} as in the previous case.
3. If i appears *after* the first w-tuple, then it is too late and fails to read v_{x-1}.

The reader processes satisfying Cases (1) and (2) are said to be *recorded* in $SLR[x]$. The function READERS in Algorithm 2 returns the ids of readers recorded in a sliding register (which is used, in particular, to update the audit set). The processes satisfying Case (2) are called *helped* in $SLR[x]$.

Regarding Case (3), we prove that after at most two failed attempts, a reader p_i receives help, or succeeds in appearing before any w-tuple in a sliding register. The algorithm also ensures that a READ operation is helped at most once. Before an attempt to write its id into $SLR[x]$, reader p_i knows, by checking if $ridx[i] \geq H[i]$, if it has already received help. If this is the case, it directly returns the corresponding value without updating $H[i]$.

4.2 Proof of Correctness

In this section, we prove Theorem 3, starting with basic properties. We show that each operation returns within $O(m+n)$ of its own steps, and then explain how to construct a sequential execution λ that contains all completed operations in an execution α, and some pending operations. Lemma 13 shows that λ preserves the real-time order between operations in α. By Lemma 14, Lemma 15, and Lemma 16, λ is a sequential execution of a m-reader, n-writer auditable register.

To proceed with the detailed proofs, fix a finite execution α of the algorithm. We start with some basic properties. Lemma 4, whose proof is in the full version, shows that that each sliding register retains the complete history of the write applied to it.

▶ **Lemma 4.** *Each process applies at most one* write *to $SLR[k]$, for any $k \geq 0$,*

The max register M stores a triple $(widx, ridx, auditset)$, where $ridx$ is a m-vector and $audiset$ a set of (process,value) pairs. Triples are ordered in increasing order of their first element. At any point in the execution, the triple stored in M is thus a triple with the largest first member written to M. We also partially order m-vectors as follows: $ridx \leq ridx'$ if and only $\forall i \in \{1, \ldots, m\}, ridx[i] \leq ridx'[i]$.

▶ **Proposition 5.** *The successive values of $M.widx$ are $0, 1, 2, \ldots$.*

In the full version, we prove the following lemma by induction on k.

▶ **Lemma 6.** *Suppose* writeMax$(k, iv, _)$ *and* writeMax$(k', iv', _)$ *are applied to M. (1) if $k = k'$ then $iv = iv'$ and, (2) if $k < k'$ then $iv \leq iv'$.*

By Proposition 5, the successive values of $M.widx$ are $0, 1, \ldots$ By Lemma 6(1), $M.ridx$ remains the same while $M.widx$ does not change. Lemma 6(2) implies that the successive vectors in $M.ridx$ are ordered and form an increasing sequence.

Let $WIDX$ denote the highest index of a sliding register to which a w-tuple has been written. That is, at the end of a finite execution α, $WIDX = k$ if and only a w-tuple was written in $SLR[k]$, and no w-tuple was written in $SLR[k']$, for any $k' > k$. When $WIDX$ is changed to $k + 1$ as a result of some process p applying write$(\mathsf{w}, _, _, _)$ to $SLR[k + 1]$, $M.widx \geq k + 1$. Indeed, before writing a w-tuple to $SLR[k + 1]$, p has read k from $M.widx$ in line 30. Observe also that when $M.widx$ is changed from k to $k + 1$ by some process p, $M.widx \geq k$. Indeed, before applying M.writeMax$(k + 1, _, _)$, p reads a sequence from $SLR[k]$ that contains a w-tuple (line 11 or line 22, p is performing a READ), or has written a w-tuple to $SLR[k]$ (line 35, p is performing a WRITE). This implies that $M.widx$ is always in $\{WIDX, WIDX - 1\}$, which shows:

▶ **Proposition 7.** *For every $x \geq 0$, if $SLR[x]$ is accessed then $SLR[x - 1]$ contains a w-tuple.*

Combining these observations with Lemma 6(2), we have:

▶ **Lemma 8.** *For some $k \geq 0$, the finite execution α can be written as either $D_0\rho_0E_0\mu_1D_1\rho_1E_1 \ldots \mu_kD_k$ or $D_0\rho_0E_0\mu_1D_1\rho_1E_1 \ldots \mu_kD_k\rho_kE_k$, where:*

- *ρ_ℓ is the first step that writes a w-tuple to $SLR[\ell]$ (applied by a writer, line 35), μ_ℓ is the step that changes $M.widx$ from $\ell - 1$ to ℓ (applied within a READ, line 26 or line 16, or a WRITE, line 38).*
- *in any configuration in D_ℓ, $M.widx = \ell = WIDX + 1$, and in any configuration in E_ℓ, $M.widx = \ell = WIDX$*

Therefore, the first step that writes a w-tuple to $SLR[x]$ is preceded by steps $\rho_0, \ldots, \rho_{x-1}$ that write w-tuples to $SLR[0], \ldots, SLR[x - 1]$, and they are never deleted (by Lemma 4).

Let x_0 be the value of the local variable lsr when READ operation op by process p_i starts.

▶ **Proposition 9.** *If op enters the repeat loop (line 12), then $x_0 < x_1 < x_2 < \ldots$ where x_k, $k > 0$, is the value read from $M.widx$ (line 18) at the beginning of the kth iteration.*

If op does not terminate in the kth iteration of the loop (line 20), after reading $(x_k, _, _)$ from M, then x_k is written to $H[i]$ (line 21) before an attempt is made to place i ahead of any w-tuple in $SLR[x_k]$. In the full version, we further prove:

▶ **Lemma 10.** *If x_k is written to $H[i]$ then $i \notin$ READERS(win), for any sequence win in any sliding register $SLR[x], x_0 < x < x_k$.*

Finally, we prove, in the full version, that *op* does not find help in $SLR[x]$, for any $x, x_0 < x < x_1$.

▶ **Lemma 11.** *$i \notin$ READERS(win), for any sequence win in a sliding register $SLR[x]$, $x_0 < x < x_1$.*

An AUDIT operation reads M once and a single sliding register. A WRITE operation applies a single writeMax and a single read to M, reads $O(m)$ registers and writes $O(1)$ registers. Since there is a linearizable implementation of max registers for $(m+n)$ processes using registers with $O(m+n)$ step complexity per operation [3], this implies that the step complexity of a WRITE or a AUDIT operation is in $O(m+n)$.

Lemma 20 (Appendix A) shows that READ operations are also wait-free, by proving that the repeat loop has at most 3 iterations. This is because a writer may place a w-tuple in at most one sliding register without detecting a concurrent READ operation and helping it.

To prove linearizability, let H be the history of READ, WRITE and AUDIT operations in the execution α. For simplicity, we assume that the values written to the register in α are unique. We start by classifying the operations in H. Each classified operation *op* is also associated with an integer $idx(op)$, which is the index of a sliding register.

For READ, we distinguish *silent*, *direct* and *helped* operations. Let *op* be a READ operation by some process p_i. We denote by x_0 the value the local variable *lsr* when *op* starts.

- *rop* is *silent* if it is not the first READ operation by p_i and it immediately returns after reading $SLR[x_0]$ (line 12). This corresponds to the case in which no new WRITE operation has occurred since the last READ by p_i. We set $idx(op) = x_0$.

If there is no $x > x_0$ for which $i \in$ READERS$(SLR[x])$, *op* is *unclassified*. In that case, note that *op* has no response in H. Otherwise, let $x_1 > x_0$ be the smallest index such that $i \in$ READERS$(SLR[x_1])$. We set $idx(op) = x_1$ and say that

- *op* is *direct* if i precedes any w-tuple in $SLR[x_1]$, and *helped* otherwise, as in that case, i appears in the helping set of the first w-tuple in $SLR[x_1]$.

A WRITE(v) operation *op* by some process p_j applies at most one write to a sliding register (line 35). *op* is *unclassified* if does not write to a sliding register. Otherwise, let x be the index of the sliding register *op* writes to. $x = idx(op)$ and we say that

- *op* is *visible* if $(\mathsf{w}, j, v, _)$ is the first w-tuple written to $SLR[x_1]$, and *hidden* otherwise.

For AUDIT, only operations that have a response in H are classified. Let *op* be an AUDIT operation that terminates. We define $idx(op) = x$, where x is the value read from $M.widx$ in the first step of *op* (line 40). *op* is *non-definitive* if there is no w-tuple in the sequence it reads from $SLR[x]$ (in line 41), and *definitive* otherwise. Indeed, once a w-tuple has been written to $SLR[x]$, the set of readers of v_{x-1} no longer changes, while READ operations may still return v_{x-1} after *op* terminates otherwise (v_{x-1} the value in the first w-tuple in $SLR[x-1]$.). In Appendix A, we prove:

▶ **Lemma 12.** *If an operation op terminates before an operation op′ starts in H', then $idx(op) \leq idx(op')$.*

We define H' that contains every completed operation of H as well as some incomplete operations, to which we add a matching response. We first discard from H every AUDIT and every silent READ invocation without a matching response, as well as every invocation of an unclassified READ and WRITE operation. We then add at the end a response for each remaining READ and WRITE operation that has no response in H. The return value of a READ operation *op* with $idx(op) = x$ is the value *val* in the first w-tuple in $(\mathsf{w}, _, val, _)$ in $SLR[x-1]$. Responses are added in arbitrary order.

A linearization $\lambda(\alpha)$ of α is defined in two steps. We first order operations in H' according to their associated index, in ascending order (rule $R0$). We then order operations with the same index. Let $B(x)$ be the set of operations $op \in H'$ such that $idx(op) = x$. These operations are ordered according to the following rules.

R1 We place first silent READ, direct READ and non-definitive AUDIT operations. They are ordered according to the order in which they apply a read (for silent READ and non-definitive AUDIT) or a write (for direct READ) to $SLR[x]$ in α.

R2 We then place helped READ in arbitrary order, followed by the definitive AUDIT operations. The definitive AUDIT are ordered according to the order in which they apply a read to $SLR[x]$ in α.

R3 We next put every hidden WRITE. They are ordered according to the order in which their write to $SLR[x]$ is applied in α.

R4 The (unique) visible WRITE is placed last.

Let λ be the linearization obtained by applying linearization rules $R0$-$R4$ to the operations in H'. In Appendix A, we prove:

▶ **Lemma 13.** *If an operation op terminates before an operation op′ starts in H', then op precedes op′ in λ.*

▶ **Lemma 14.** *If READ operation op in H' returns v, then v is the value written by the last WRITE that precedes op in λ, or the initial value v_0 if there is no such WRITE.*

Finally, we prove (in Appendix A) that a pair (p, v) is in the set returned by an AUDIT operation *op* if and only if there is a READ by p returning v precedes *op*.

▶ **Lemma 15.** *Let aop be an AUDIT operation in H' that returns A. If there is a READ operation rop by process p_j returning v and that precedes aop in λ, $(j, v) \in A$.*

▶ **Lemma 16.** *Let aop be an AUDIT operation in H' whose response contains (j, v). There exists a READ operation by p_j returning v that precedes aop in λ.*

5 Auditable LL/SC from $2n$-Sliding Register

We show how to adapt our auditable register algorithm to implement an auditable LL/SC object for n processes, and an arbitrary number of auditors. An AUDIT operation returns a set of pairs (p, v), representing the values returned by the processes by the LL operations preceding the AUDIT. The implementation uses $2n$-sliding registers, showing:

▶ **Theorem 17.** *There is a wait-free, linearizable implementation of an n-process auditable LL/SC object from $2n$-sliding registers and standard registers with $O(n)$ step complexity per operation.*

Algorithm 3 is essentially the same as Algorithm 1. LL operations are identified with READ operations, and SC with WRITE. At the end of a finite execution α of Algorithm 1, the sequences stored in the sliding registers in the array SLR indicate, for each x, which is the xth value v_x held in the auditable register, and which processes write or read this value. Specifically, given the sequence win_x stored in $SLR[x]$, the first w-tuple (w, j, v, h) in win_x indicates that $v_x = v$, and its writer is p_j. The readers of v_{x-1} are the processes p_i, where $i \in h$, or precedes (w, j, v, h) in win_x. Each other w-tuple $(\mathsf{w}, j', v', _)$ in win_x

Algorithm 3 n-process LL/SC with auditable LL.

```
1: shared variables
2:     M, H[1..n]: max register and helping array of n SWMR register, initialized as in Algorithm 1
3:     SLR[−1, 0, ...]: unbounded array of 2n-sliding registers, initialized as in Algorithm 1
4: local variables
5:     lsr, val, as in Algorithm 1
6: function LL( )                                   ▷ identical to READ in Algorithm 1
7: function SC(v)                                   ▷ code for process p_i, i ∈ {1, ..., n}
8:     widx, ridx, auditset ← M.read(); to_help ← ∅
9:     if widx > lsr then return false              ▷ a successful SC happened since p_i's last LL
10:    for all i ∈ {1, ..., m} do aidx_j ← H[j].read()   ▷ help ongoing LL operations
11:        if ridx[j] < aidx_j then                 ▷ an LL by p_j may need help
12:            window ← SLR[aidx_j].read()
13:            if j ∉ READERS(window) then to_help ← to_help ∪ {j}
14:    SLR[widx].write(w, j, v, to_help);           ▷ announce new SC
15:    window ← SLR[widx].read(); val ← GETVALUE(widx − 1)
16:    for each j ∈ READERS(window) do ridx[j] ← widx; auditset ← auditset ∪ {(j, val)}
17:    M.writeMax(widx + 1, ridx, auditset);
18:    if (w, j, v, to_help) is the 1st w-tuple in window then return true else return false
19: function AUDIT,GETVALUE, READERS                ▷ as in Algorithm 1
```

corresponds to an *invisible* write operation, whose input v' is never read. We may think as these operations as *unsuccessful*, in the sense that they fail to change the value of the auditable register, being immediately overwritten by another write.

Alternatively, we may think of the sequences in SLR as a trace of an execution β of an LL/SC object implementation by identifying reads with LL and writes with SC. Again, the xth value held by the object in β is v_x, the value in the first w-tuple (w, j, v, h) in win_x. Each process p_i, with $p_i \in h$ or preceding this tuple has an LL that returns v_{x-1}. The first w-tuple (w, j, v, h) indicates a successful SC(v) by process p_j, that changes the object from v_{x-1} to $v_x = v$. And every following tuple $(\mathsf{w}, j', v', _)$ marks an unsuccessful SC(v') by process $p_{j'}$. Recall that, per its specification, an SC is successful if and only if it is preceded by an LL by the same process, without any successful SC operation between them. In particular, β is a valid sequential execution if (1) in each sequence win_x, the first w-tuple (w, j, v, h) is after p_j's LL and (2) each SC(v') operation corresponding to a following $(\mathsf{w}, j', v', _)$ tuple is after SC(v) in β.

The code of an LL operation is the same as for a READ operation in Algorithm 1, while AUDIT and the auxiliary functions GETVALUES and READERS are identical. SC operations follow the code of WRITE, with two additions (line 9 and line 18) highlighted in gray.

To maintain property (1), an SC operation *op* by process p_i should be prevented from writing a w-tuple to a sliding register $SLR[x]$ in which p_i 's last LL is not recorded. In Algorithm 1, the local variable lsr of a process p_i is the highest index of a sliding register that keeps track of p_i's last READ, and hence here, $SLR[lsr]$ records p_i's last LL. Therefore, before writing to $SLR[widx]$ (in line 14), p_i checks that $lsr = widx$. If this is not the case, a successful SC has occurred since p_i's last LL, and *op* may immediately return *false* (line 9). The other addition is the return statement at line 18: *true* is returned if p_i's w-tuple is the first in $SLR[widx]$, and *false* otherwise.

For property (2), the linearization rules are slightly modified. In Algorithm 1, WRITE operations recorded in the same sliding register $SLR[x]$ are linearized in the reverse order of their corresponding w-tuple appearance in the sequence stored in $SLR[x]$. Here, we do the

opposite, linearizing SC operations in the same order their corresponding w-tuple appear in $SLR[x]$. Only the first w-tuple represents a successful SC, which is aligned with the return statement of line 18.

Finally, as each process may write twice to a sliding register, once in an LL operation, and once in a SC operation, SLR is an array of $2n$-sliding registers. The code of AUDIT may easily be adapted to report which process has successfully stored which value instead of or in complement to values returned by LL operations to the processes.

The code of AUDIT is the same in Algorithm 1 and Algorithm 3, and the same code is shared by READ and LL operations. For SC, the additional statements (line 9 and line 18) in the code do not affect termination or step-complexity. Therefore, the step complexity of LL, SC, and AUDIT is $O(n)$.

Fix a finite *well-formed* execution β, in which each process alternates LL and SC operations. Thanks to the similarities in the code, a linearization $\mu(\beta)$ of β can be obtained from a linearization $\lambda(\alpha)$ of an execution α of the auditable register implementation induced by β. α is constructed as follows. We introduce a new class for SC operations that terminate immediately after reading $M.widx$ in line 9. Those operations are said to be *silent* (similarly to silent READ operations). We next extract from β an execution α of Algorithm 1 by removing all steps applied by silent SC, and replacing each invocation of SC and LL by and invocation of WRITE and READ, respectively, with the same input. α is a valid execution of Algorithm 1, as besides early termination for SC (line 9), which we dispose of by removing steps of silent SC operations, the code of SC and LL is the same as the code of WRITE and READ, respectively, and the code for AUDIT is identical. We explain in Appendix B how a linearization $\mu(\beta)$ can be inferred from the linearization $\lambda(\alpha)$ of α.

6 Immediate Deny List from Auditable Registers

A *Deny List* [11] is used to control resources, by having a set of managers maintain a list of which users are unauthorized to access which resources. To access a resource, a user must prove that the corresponding user-resource pair has not been added to the Deny List. The managers have to agree on the set of process-resource pairs in the Deny List. A Deny List has an *anti-flickering* property that ensures that there are no transient periods: once access is disallowed, it is never allowed again.

More formally, a *deny list* object over a set of resources S supports three operations, for $x \in S$: APPEND(x), PROVE(x) which returns a Boolean value, and READ(), which returns a set of process-resource (p, x) pairs. A PROVE(x) that returns *false* is *invalid*; otherwise, it is *valid*. The intuition is that an APPEND(x) revokes the authorization to access a resource x to all processes. A valid PROVE(x) by process p indicates that p is authorized to access x. The set of processes that can invoke APPEND is called the *managers*, and those that can invoke PROVE are called the *provers*. These sets of processes are predefined and static. The property *termination* requires that the operations PROVE, APPEND, and READ return within a finite number of steps.

In the original definition [11], an APPEND can be *successful* or *unsuccessful*. Their sequential specification in the *anti-flickering flavor* of a deny list is as follows:

Append progress: Only a finite number of APPEND(x) operations are unsuccessful; that is, APPEND(x) is eventually successful.

Prove progress: After a successful APPEND(x) operation, only a finite number of PROVE(x) operations can be valid; that is, PROVE(x) is eventually invalid after a successful APPEND(x).

Prove validity: A PROVE(x) operation is invalid only if a successful APPEND(x) operation appears before it.

Prove anti-flickering: If a PROVE(x) operation *op* is invalid, then all following PROVE(x) operations are invalid.

Read validity: The set of object-process pairs returned by a READ() operation includes exactly all preceding valid PROVE(x) operations and the processes that invoked them.

We consider a stronger version, called *immediate* deny list, where all APPEND(x) are successful, and where all PROVE(x) operations that follow an APPEND(x) are invalid. The sequential specification for the *immediate* deny list is therefore as follows:

Strong prove validity: A PROVE(x) operation is invalid if and only if an APPEND(x) operation appears before it.

Read validity: The set of object-process pairs returned by a READ() operation includes exactly all preceding valid PROVE(x) operations and the processes that invoked them.

An *Immediate Deny List* guarantees all the properties of the *Anti-flickering Deny List*: Since all APPEND(x) are successful, we trivially guarantee *Append progress*. Our *Strong prove validity* includes the *Prove validity* and *Prove progress* where the number of PROVE(x) that can be valid after an APPEND(x) is 0. *Prove anti-flickering* is trivially implied by our *Strong prove validity* property.

It is already known that the consensus number of an *Anti-flickering Deny List* object where n processes can both do APPEND(x) and PROVE(x), denoted anti-flickering n-deny list, is n [11]. In particular, they show how to solve consensus among n processes using this object with a single resource. Thus, an Anti-flickering n-deny list has at least consensus number n, and therefore, the consensus number of an immediate n-deny list is also at least n.

They also present an algorithm to show that the consensus number of the anti-flickering n-deny list over a set of resources S is at most n. In the following, we show how to implement (Algorithm 4) an immediate-n-deny list object using $(1, n-1)$-auditable registers, which have consensus number n as proved in Sections 3 and 4. This proves that the immediate n-deny list has consensus number at most n. Our algorithm provides a somewhat simpler proof for their upper bound. We present the algorithm for a single resource. The generalized version can be easily built thanks to the locality property of linearizability, and by implementing the READ() on the set of resource $\mathcal{S}$ using classical techniques to implement a snapshot by applying the READ() on each $x \in \mathcal{S}$.

Algorithm 4 implements an immediate n-deny list for a resource x. It uses a vector $AR_x[1 \dots n]$ of binary auditable registers, initially *true*. $AR_x[i]$ is written by process p_i and read by all other processes. When a process p_i wants to perform APPEND(x), it writes false in its auditable register $AR_x[i]$ and locally stores the information that it did an APPEND.

To perform PROVE(x), a process simply checks if it previously did an APPEND(x), and if not, it reads all the auditable registers (but its own) to check if some process wrote into one such register. If none of the previous conditions happen, then it can return *true*.

The READ() repeatedly audits all the registers to collect the processes that have executed a valid PROVE(x). These are the ones that read *true* in all registers. To obtain a snapshot, the READ() terminates when two consecutive collects returns the same set. Appendix C presents proof sketch of the correctness of the algorithm.

7 Conclusions and Future Work

In this work, we extended the concept of auditability from single-writer registers to more general shared objects. We start by providing a rigorous characterization of the synchronization power required to support auditable multi-writer registers. Our results establish a

Algorithm 4 Immediate n deny-list object from auditable registers, code for process p_i.

Shared objects
$AR_x[1 \dots n]$ is a vector of $(1, n-1)$-auditable registers, initially *true*
local variables
$append_x$ is a boolean initially *false*, that is set to *true* when p_i does APPEND(x)
function APPEND(x)
$AR_x[i]$.write(*false*); $append_x \leftarrow$ *true* **return**
function PROVE(x)
if $append_x$ **then return** *false*;
for all $j \in \{1, \dots, n\}$ with $j \neq i$ **do**
$b \leftarrow AR_x$.read(); **if** $\neg b$ **then return** *false*
return *true*
function READ( ) ▷ audit all registers until you get a successful double collect
$c2 \leftarrow \emptyset$
repeat
$c1 \leftarrow c2$; $c2 \leftarrow \emptyset$
for all $j \in \{1, \dots, n\}$ **do** $a_j \leftarrow AR_x[j]$.audit()
$c2 \leftarrow \{(q, x) : \forall j \neq q, (q, true) \in a_j\}$
until $c1 = c2$
return $c2$

tight bound on the consensus number necessary for achieving auditability, demonstrating the feasibility of implementing auditable storage mechanisms.

Looking ahead, there are several promising directions for future research. First, extending auditability to a broader range of shared objects beyond registers and LL/SC remains an open challenge. Second, investigating the impact of adversarial behavior on auditable implementations could lead to more robust security guarantees. Finally, exploring efficient, scalable implementations of auditable objects in real-world distributed storage systems could bridge the gap between theoretical feasibility and practical deployment. In particular, while sliding registers are not supported in hardware (to the best of our knowledge), they bear resemblance to *shift registers*. This hardware object [16] holds the last w bits "shifted in" the register and has a functionality similar to a sliding register; the consensus number of shift registers is w [2], Although they cannot directly replace sliding registers in our implementations, our algorithmic insights might be leveraged to develop other algorithms that employ shift registers.

References

1 Sara Alhajaili and Arshad Jhumka. Auditability: An approach to ease debugging of reliable distributed systems. In *IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 227–2278, 2019. doi:10.1109/PRDC47002.2019.00053.

2 James Aspnes. The consensus number of a shift register equals its width, 2025. doi:10.48550/arXiv.2505.01691.

3 James Aspnes, Hagit Attiya, and Keren Censor-Hillel. Polylogarithmic concurrent data structures from monotone circuits. *Journal of the ACM*, 59(1), March 2012. doi:10.1145/2108242.2108244.

4 Hagit Attiya, Antonio Fernández Anta, Alessia Milani, Alexandre Rapetti, and Corentin Travers. Auditing without leaks despite curiosity. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 455–465. ACM, 2025. doi:10.1145/3732772.3733516.

5 Hagit Attiya, Antonio Fernández Anta, Alessia Milani, Alexandre Rapetti, and Corentin Travers. Auditable shared objects: From registers to synchronization primitives, 2025. arXiv:2508.14506.

6 Hagit Attiya, Antonella Del Pozzo, Alessia Milani, Ulysse Pavloff, and Alexandre Rapetti. The synchronization power of auditable registers. In *27th International Conference on Principles of Distributed Systems OPODIS*, volume 286 of *LIPIcs*, pages 4:1–4:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.OPODIS.2023.4.

7 Richard A Becker and John M Chambers. Auditing of data analyses. *SIAM Journal on Scientific and Statistical Computing*, 9(4):747–760, 1988.

8 Vinicius Vielmo Cogo and Alysson Bessani. Brief announcement: Auditable register emulations. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 53:1–53:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.53.

9 Antonella Del Pozzo, Alessia Milani, and Alexandre Rapetti. Byzantine auditable atomic register with optimal resilience. In *41st International Symposium on Reliable Distributed Systems (SRDS)*, pages 121–132. IEEE Computer Society, 2022. doi:10.1109/SRDS55811.2022.00020.

10 Faith Ellen, Panagiota Fatourou, Eleftherios Kosmas, Alessia Milani, and Corentin Travers. Universal constructions that ensure disjoint-access parallelism and wait-freedom. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 115–124, 2012. doi:10.1145/2332432.2332457.

11 Davide Frey, Mathieu Gestin, and Michel Raynal. The synchronization power (consensus number) of access-control objects: the case of allowlist and denylist. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 21:1–21:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.21.

12 Boris Glavic et al. Data provenance. *Foundations and Trends® in Databases*, 9(3-4):209–441, 2021. doi:10.1561/1900000068.

13 Reza Hajisheykhi, Mohammad Roohitavaf, and Sandeep S. Kulkarni. Bounded auditable restoration of distributed systems. *IEEE Transactions on Computers*, 66(2):240–255, 2017. doi:10.1109/TC.2016.2595578.

14 Maurice P Herlihy and Jeannette M Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

15 Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015. doi:10.1109/MC.2015.33.

16 Intel. x86 assembly/shift and rotate. URL: https://en.wikibooks.org/wiki/X86_Assembly/Shift_and_Rotate.

17 Eric H Jensen, Gary W Hagensen, and Jeffrey M Broughton. A new approach to exclusive data access in shared memory multiprocessors. Technical report, Technical Report UCRL-97663, Lawrence Livermore National Laboratory, 1987.

18 Henry M Levy. *Capability-based computer systems*. Digital Press, 2014.

19 Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. A simple object that spans the whole consensus hierarchy. *Parallel Process. Lett.*, 28(2):1850006:1–1850006:9, 2018. doi:10.1142/S0129626418500068.

20 Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998. doi:10.1016/S0065-2458(08)60206-5.

21 Ravi S. Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994. doi:10.1109/35.312842.

A Additional Lemmas and Proofs for Section 4

We present in this appendix missing proof and additional lemmas for Section 4. Omitted proofs can be found in the full version.

The function READERS extracts from a sequence read from some sliding register $SLR[x]$ a set of processes. This set does not change once a w-tuple has been written to $SLR[x]$.

▶ **Proposition 18.** *If win and win′ are two sequences read from $SLR[x]$ after $SLR[x]$.write(w, _, _, _) has been applied, then READERS(win) = READERS(win').*

▶ **Lemma 19.** *$M.ridx[i] \geq x$ if and only if $SLR[x]$ holds a sequence win such that $i \in$READERS(win).*

▶ **Lemma 20.** *The step complexity of a READ operation is in $O(m+n)$.*

▶ **Lemma 12.** *If an operation op terminates before an operation op′ starts in H', then $idx(op) \leq idx(op')$.*

Proof. Let $x = idx(op)$ and $x' = idx(op')$, and let p_i and $p_{i'}$ be the processes that perform op and op' respectively.

We first prove that when op terminates, $M.widx \geq x$. As $M.widx$ is increasing (Proposition 5), it is enough to show that $M.widx \geq x$ in a configuration in the execution interval of op. If op is a WRITE or an AUDIT, op writes to $SLR[x]$ (line 35) or reads from $SLR[x]$ (line 41) after reading x from $M.widx$ (line 30 or line 40).

If op is a direct READ, p_i writes i to $SLR[x]$ (line 22) after having read x from $M.widx$ (line 18). If op is helped, $M.ridx[i] = x$ when M is read at the beginning of some iteration of the loop (line 18). Therefore $M.widx > x$, as the writeMax that changes $M.ridx[i]$ to x also set $M.widx$ to $x+1$ (lines 15-16, lines 25-26, or lines 37-38). If op is silent, it is preceded by direct READ op'' with the same index x, and hence $M.widx \geq x$ already when op'' terminates.

We next examine several cases according to the type of op'.

- op' is a silent READ. In op', $SLR[x']$ is read by $p_{i'}$ and does not contain a w-tuple. Hence, before op starts, $WIDX = x' - 1$ (the largest index of a sliding register to which a w-tuple has been written), and therefore, by Lemma 8, $M.widx \leq x' - 1$, from which we have $x \leq x' - 1$, as $x \leq M.widx$ when op terminates.
- op' is a direct or a helped READ. Let x_0 be the value of the local variable lsr when op' starts. op' does not terminate before reading some value x_1 from $M.widx$ (line 18) in the first iteration of the repeat loop. By Lemma 11, for every y, $x_0 < y < x_1$, the helping set of the first w-tuple in $SLR[y]$ does not contains i'. Therefore, $x_1 \leq x' = idx(op')$ and hence $x \leq x'$ as $x \leq M.widx$ before op' starts.
- op' is a WRITE or an AUDIT. $p_{i'}$ reads x' from $M.widx$ (line 30 and line 40, respectively). As $x \leq M.widx$ when op terminates, $x \leq x'$. ◀

To show real-time order is preserved, we first prove the following facts about the precedence of operations with the same sequence number.

▶ **Lemma 21.** *Let op, op′ be two operations in H' with $idx(op) = idx(op') = x$. Let $x \geq 0$.*

1. *If op is a silent READ, a direct READ, or a non-definitive AUDIT, and op′ is a helped READ, a definitive AUDIT, or a WRITE operation, then op′ does not precede op.*
2. *If op is any operation and op′ is a WRITE, then op′ does not precede op.*

▶ **Lemma 22.** *Let op be a helped* READ *operation in H' with $idx(op) = x$. The first write of a w-tuple to $SLR[x]$ happens during the execution interval of op.*

▶ **Lemma 13.** *If an operation op terminates before an operation op' starts in H', then op precedes op' in λ.*

Proof. By Lemma 12, $idx(op) \leq idx(op')$. If $idx(op) < idx(op')$, op is before op' in λ by rule $R0$. We assume in the following that $idx(op) = idx(op') = x$.

- If op' is a silent READ, a direct READ, or non-definitive AUDIT, it follows from Lemma 21(1) that op also falls into this category. Therefore, op and op' are both ordered in λ using rule $R1$. They are ordered according to the order in which a step in their execution interval occurs in α. Hence, op precedes op' in α implies that op precedes op' in λ.
- If op' is a helped READ or a definitive AUDIT, it follows from Lemma 21(1) and Lemma 21(2) that op also falls into this category, or is a silent or direct READ, or a non-definitive AUDIT. In the latter case, op is ordered in λ according to rule $R1$, and op', rule $R2$, from which we have that op precedes op' in λ.
  In the former case, as op and op' are both helped READ, their execution interval intersect (Lemma 22) and thus op cannot precedes op'. If op and op' are both definitive AUDIT, they are placed according to the order in which they apply a read to $SLR[x]$ in α. Hence op is before op' in λ. If op is an helped READ and op' a definitive AUDIT, op is placed before op' in λ by rule $R2$. The last case remaining is op being a definitive AUDIT and op', a helped READ. As op is a definitive AUDIT, it sees a w-tuple mark in $SLR[x]$, but the step in which the first such tuple is written to $SLR[x]$ is in the execution interval of op' (Lemma 22). Therefore op cannot terminate before op' starts.
- If op' is a WRITE visible or hidden, Lemma 21(2) implies that op cannot be a WRITE. Therefore, op is placed according to rule $R1$ or $R2$, and sthus precedes op' which is placed after, according to $R3$ or $R4$. ◀

▶ **Lemma 14.** *If* READ *operation op in H' returns v, then v is the value written by the last* WRITE *that precedes op in λ, or the initial value v_0 if there is no such* WRITE.

Proof. Let $x = idx(op)$, and let p_j be the process that performs op.

We first consider the case $x > 0$. If op is helped or direct, the value returned by op is the value v in the first w-tuple $(\mathsf{w}, i, v, _)$ stored in $SLR[x-1]$ (lines 45). Note that i and v are well defined, as when p_j reads from or writes to $SLR[x]$, a w-tuple has already be written to $SLR[s-1]$ (Proposition 7). Let wop be the operation p_i is performing when it writes $(\mathsf{w}, i, v, _)$ to $SLR[x-1]$. By definition, $idx(op) = x - 1$, and wop is a visible WRITE. Therefore, among the operations with index $x - 1$, wop is placed last in λ (rule $R4$), and there is no other WRITE operation between wop and op (every WRITE operation with index $\geq x$ is after op in λ.)

If rop is silent, it is preceded by a direct READ op' by the same process with the same index x. By the linearization rule $R1$, there is no WRITE operation between op' and op in λ. Also, op and op' return the same value. By the same reasoning as above, it follows that op returns the input value of the last WRITE operation that precedes it in λ.

If $x = 0$, op returns the initial value v_0. Indeed, $SLR[-1]$ is initialized with a sequence that contains a single tuple $(\mathsf{w}, i_0, v_0, \emptyset)$. There is no WRITE that precedes op in λ, as for every WRITE operation wop, $idx(wop) \geq 0$. ◀

▶ **Lemma 23.** *If $i \in$ READERS$(SLR[x])$ then there exists a* READ *operation op in H' by process p_i with $idx(op) = x$.*

Proof. Suppose that i precedes any w-tuple in $SLR[x]$. p_i writes i to $SLR[x]$ (line 22) while performing the kth iteration of the repeat loop (lines 17-27) in some READ operation op. Let x_0 be the value of lsr when op starts, and let $x_1 < \ldots < x_k$ be the value read from $M.widx$ (line 18) in the first k iterations of the loop. Note that $x = x_k$. As before writing i to $SLR[x_k = x]$, $H[i]$ is changed to x_k (line 21), it follows from Lemma 10 that for every $x', x_0 < x' < x_k$, $i \notin$ READERS$(SLR[x'])$. As $i \in$ READERS$(SLR[x_k])$, $x_k = \min\{x' > x_0 : i \in$READERS$(SLR[x'])\}$ and therefore by definition $x = x_k = sn(op)$.

Suppose now that i is in the helping set of the first w-tuple $(\mathsf{w}, j, _, h)$ written to $SLR[x]$. Before writing $(\mathsf{w}, j, _, h)$ (line 35), p_j reads x from $M.widx$, y from $M.ridx[i]$ (line 30) and x_ℓ from $H[i]$ (line 31). As i is placed into to_help, $y < x_\ell$.

$H[i]$ is changed by p_i (line 21) in an iteration of the repeat loop while performing some READ operation op. Let x_0 be the value of lsr when op starts, and p_i is performing iteration ℓ when it writes x_ℓ to $H[i]$. By the code, x_ℓ is the value read from $M.widx$ in that iteration, and by Lemma 10, $i \notin$ READERS$(SLR[x'])$, for every $x', x_0 < x' < x_\ell$.

To summarize, starting from $i \in$READERS$(SLR[x])$, we have shown that there exists a READ operation op by p_i that starts with $lsr = x_0$, and that for every $x', x_0 < x' < x(= x_\ell)$, $i \notin$READERS$(SLR[x'])$. By definition, $idx(op) = x$. ◀

▶ **Lemma 15.** *Let aop be an* AUDIT *operation in H' that returns A. If there is a* READ *operation rop by process p_j returning v and that precedes aop in λ, $(j, v) \in A$.*

Proof. By linearization rules $R0$-$R2$, $idx(rop) = x \leq idx(aop)$. Let p_i be the process that performs the AUDIT operation aop.

If rop is silent, it is preceded by a direct READ operation by the same process, with the same output value v and the same index x. In the following, we thus assume that rop is direct or helped. In $SLR[x]$, j is written before any w-tuple, or the first w-tuple written to $SLR[x]$ has an helping set h containing i. Note that v is the value of the first w-tuple in $SLR[x-1]$.

Let us assume that $x < idx(aop) = x'$. In aop, p_i reads x' from $M.widx$ and a set as' from $M.auditset$ (line 40). Before this step, M is changed from x to $x+1$ (in line 16, line 26 or line 38), after a win containing a w-tuple is read from $SLR[x]$. Hence, $j \in$ READERS(win), and therefore (j, v) is added to the audit set as written together with $x+1$ to M. By the code (line 15, line 25 or line 37 and line 42), we have that $as \subseteq as' \subseteq A$.

We now assume that $x = idx(aop)$. If aop is a non-definitive AUDIT, no w-tuple has been written to $SLR[x]$ when p_i reads $SLR[x]$. As rop precedes aop in λ, there are both placed in λ following rule $R1$ and therefore p_j writes j to $SLR[x]$ before the sliding register is read in aop, from which it follows that $(j, v) \in A$ (line 42). Otherwise, aop is a definitive AUDIT, which means that the value win reads from $SLR[x]$ (line 42) contains the first w-tuple $(\mathsf{w}, _, _, h)$ written to $SLR[x]$. As j precedes this tuple or $i \in h$, $j \in$ READERS(win) and therefore $(j, v) \in A$. ◀

▶ **Lemma 16.** *Let aop be an* AUDIT *operation in H' whose response contains (j, v). There exists a* READ *operation by p_j returning v that precedes aop in λ.*

Proof. Let p_i be the process that performs aop, and let A be the set of pairs (process,value) returned by this operation. As (j, v) is in A, there exists x such that $i \in$READERS$(SLR[x])$ and v is the value in the first w-tuple $(\mathsf{w}, _, v_)$ written to $SLR[x-1]$. Indeed, pair (j, v) is inserted to A by p_i after reading $SLR[x]$, if $x = idx(aop)$ (line 42), or when $M.widx$ is changed to $x+1$ (lines 15-16, lines 25-26 or lines 37-38) if $x < idx(aop)$.

By Lemma 23 shows there is a READ operation op by process p_i with $idx(op) = x$. This READ returns v, which is the value of the first w-tuple written to $SLR[x-1]$. It remains to prove that op precedes aop in λ. If $x < idx(aop)$, op precedes aop in λ (rule $R0$). Otherwise, $x = idx(aop)$. If aop is definitive, it is linearized after rop by rules $R1$ and $R2$. Else, aop is non-definitive, and j is before any w-tuple in $SLR[x]$. j is also written to $SLR[x]$ before before $SLR[x]$ is read by p_i. op is hence direct, and linearized before aop by rule $R1$. ◀

B Proof Sketch for Section 5

Recall that β is an execution of the auditable LL/SC implementation (Algorithm 3) and α an execution of the auditable register implementation induced from β. We build a linearization $\mu(\beta)$ of β from the linearization $\lambda(\alpha)$ of α as follows:

1. WRITE operations with the same index x are reordered, by applying first rule $R4$ and then rule $R3$. Hence, the WRITE operation corresponding to the first w-tuple in $SLR[x]$ is placed first, and then the WRITEs corresponding to the other w-tuples follow, in arbitrary order. The resulting sequence $\lambda'(\alpha)$ is no longer a valid linearization of an auditable register, but still extends the real-time order among operations, as WRITEs with the same index are concurrent (Lemma 21(2)). Observe that in $\lambda'(\alpha)$, as in $\lambda(\alpha)$, WRITEs with the same index form a contiguous block, denoted W_x. A READ operation no longer returns the input of the last preceding WRITE, but rather the input v_x of the first WRITE in the last block W_x that precedes it.
2. We revert to LL/SC operations, by replacing in $\lambda'(\alpha)$ each READ and WRITE operation by the LL or SC operation it originated from. This results in a partial linearization $\mu'(\beta)$ of β, missing silent SC operations. In $\mu'(\beta)$, an LL operation op returns the input v_x of the first SC operation sop in the last block W_x that precedes it. Every SC operation in W_x writes a w-tuple to $SLR[x]$, sop being the first to do so. Hence, except sop, each of them returns false (line 18). Consequently, op returns the input of the last preceding successful SC.
   If p_i has a SC operation in W_x, its matching preceding LL operation op is recorded in $SLR[x]$ (line 9) and therefore, has index $idx(op) = x$. By rules $R1$-$R2$, op is before W_x, and after any operation op' with index $idx(op') < x$ (rule $R0$). Only the first SC operation sop in W_x returns true, and indeed, there is no SC between sop and its last preceding LL. Each other SC operation in W_x is unsuccessful, in agreement with the fact that it preceded by the successful SC sop, with no LL operation in between.
   LL and AUDIT are ordered in $\mu'(\beta)$ as their corresponding READ and AUDIT in $\lambda(\alpha)$, and each LL returns the same value as its corresponding READ. Hence, a pair (p, v) is included in the response of an AUDIT operation op if and only if there is an LL by p that returns v before op in $\mu'(\beta)$.
3. Finally, we bring back the silent SC operations that were removed from β to form a full linearization $\mu(\beta)$. Each silent SC operation op that returns is unsuccessful and therefore can be inserted in $\mu'(\beta)$ without affecting the outcome of other SC or LL operations, as long as there is a successful SC between the matching LL op' by the same process and op. Let x be the value read by op from $M.widx$. We insert op in $\mu'(\beta)$ within the operations whose index is x and after every operation that terminates before op starts. As $x > x'$, op is placed after the operations with index x'. In particular, op follows the successful SC operation sop that writes the first w-tuple to $SLR[x]$, which in turn follows op'. This can be done while preserving the real-time order, as when $x > x'$ is read from $M.widx$ in op, a w-tuple has already been written to $SRL[x']$ (Lemma 8).

C Proof Sketch for Section 6

To prove the correctness of Algorithm 4, we start by noting that all operations terminate within a finite number of their own steps. This is obvious for APPEND and PROVE, relying on wait-freedom of the encapsulated auditable register operations. For a READ operation, note that the sets it returns are monotonically contained in each other; the maximal set is the one containing all process-object pairs. Furthermore, if two sets are equal, the operation terminates. Thus, the loop can be repeated at most n times.

We next prove that Algorithm 4 implements a linearizable immediate n-deny list. First, for all $i \in \{1, \ldots, n\}$, we linearize all the APPEND(x) in the step corresponding to the write to the auditable register. APPEND(x) that have not reached this step are discarded.

A PROVE(x) operation by p_i that returns *false* and that does not return at line 6 is linearized at the last read primitive p_i applies to an auditable register $AR_x[j]$. A PROVE(x) that returns at line 6 is linearized at its invocation. A READ is linearized at the last audit of the second last loop (i.e., the last audit before c_1 is set for the last time). We remove incomplete READ.

PROVE(x) operations that return *true* are inserted at the beginning before the first APPEND(x), according to their real-time order. In particular, we pick one by one in the order of their last step (i.e., the last read from an auditable register), the ones that happen earlier first), denoted s. We put PROVE(x) immediately before the first READ() whose linearization point follows s or immediately the first APPEND(x) if such READ() does not exist.

Strong prove validity: A PROVE(x) by a process p returns *false*, either if p previously performed an APPEND(x) or if it read *false* in one auditable register. In both cases, there is an APPEND(x) in the set of operations we linearize and it is linearized before the PROVE(x) according to our rules. The only if part is immediate since we linearize all PROVE(x) that return *true* before the first APPEND(x).

▶ **Lemma 24.** *The set of object-process pairs returned by a* READ() *operation includes exactly all preceding valid* PROVE(x) *operations and the processes that invoked them.*

Proof. A $op =$ PROVE(x) that returns *true* is linearized before the first audit, namely op', whose linearization point follows the last read primitive applied by op to an auditable register. After the step where it is linearized op' executes a second loop where it audits all the auditable registers. At that point op is added in c_2. By the definition of the linearization point of op', op is in the set returned by op'.

On the other hand, suppose that a pair $(q, true)$ is in the set c_2 returned by a READ() by p. Then, there is a PROVE(x) by q that read *true* in all the auditable registers (but $AR_x[q]$) before p start the execution of the last loop. By a simple inspection of the pseudo code it is easy to see that op returns *true*. Our linearization rule completes the proof. ◀

Finally, we show that our linearization respects the *real-time order* of operations. All operations but the PROVE(x) that return *true* are linearized in a point of their execution. A PROVE(x) operations that returns *true* is linearized before the first APPEND(x) in the linearization. As it reads *true* in all low-level auditable registers, there is no APPEND(x) that completes before this PROVE(x) is invoked, otherwise one of the low-level read would have returned *false*. If PROVE(x) op completes before a READ() op' starts, then the last step of op precedes the linearization point of op'. Thus, op is linearized before op'. Similarly if op is invoked after op' completes, then the linearization point of op' precedes the last step of op which is then linearized after op'. This concludes the proof since PROVE(x) are linearized in the order of their last step.

Distributed Download from an External Data Source in Byzantine Majority Settings

John Augustine
Indian Institute of Technology Madras, India

Soumyottam Chatterjee
CISPA Helmholtz Center for Information Security, Saarbrückem, Germany

Valerie King
University of Victoria, Canada

Manish Kumar
Indian Institute of Technology Madras, India

Shachar Meir
Weizmann Institute of Science, Rehovot, Israel

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Abstract

We consider the Download problem in the Data Retrieval Model, introduced in DISC'24, where a distributed set of peers, some of which may be Byzantine, seek to learn n bits of data stored at a trustworthy external data source. Each bit of data can be learned by a peer either through a direct and *costly* query of the source or through other peers that have already learned it; the goal is to design a collaborative protocol that reduces the query complexity defined as the maximum number of bits queried by any honest peer.

We begin with a randomized protocol for the Download problem that achieves optimal query complexity, up to a logarithmic factor. For a stronger "dynamic" adversary that can change the set of Byzantine peers from one round to the next, we achieve optimality (within log factors) for both query complexity (in expectation) and time complexity, but with larger messages. In broadcast communication, where all peers (including Byzantine peers) are required to send the same message to all peers, we achieve (up to log factors) an optimal trade-off between query complexity, time complexity, and message size with the dynamic adversary. All of our protocols can tolerate any constant fraction $\beta < 1$ of Byzantine peers.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine Fault Tolerance, Blockchain Oracle, Data Retrieval Model, Distributed Download

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.9

Related Version *Full Version*: `https://arxiv.org/abs/2412.19649` [5]

Funding *John Augustine*: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
Soumyottam Chatterjee: Work done while at IIT Madras, supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
Manish Kumar: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
David Peleg: Venky Harinarayanan and Anand Rajaraman Visiting Chair Professor. The funds from this professorship enabled exchange visits between IIT Madras, India, and the Weizmann Institute of Science, Israel.



© John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Almost 45 years ago, the Byzantine Generals Problem, which launched the study of Byzantine agreement, was posed as follows:

> "We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action." [18]

One component of the problem is that each general v must first evaluate the situation and make observations about the enemy that are used to guide its choice of action $x(v)$ ("attack" or "retreat"). The common formal definition of the agreement problem, as given in the literature, abstracts away the observation stage, and considers the action choice $x(v)$ of each general v as its *input bit* (allowing those bits to differ even among the honest generals).

The Data Retrieval (DR) model [4] "zooms in" on the stage of collecting the observations. Suppose that there are n questions of interest about the enemy, which the generals are required to answer to reach a correct action choice. Denote the answers to these questions (conveniently assumed to be Boolean) by the *observation bits* $b_1, \ldots, b_n$. Making all n observations and learning the entire *observations vector* $\bar{b}$ is possible but expensive for any single general v. The DR model addresses this by allowing the generals to collaborate in order to make all n observations and share the bits b_i correctly. Consequently, each general v can compute its choice $x(v)$ based on the entire observations vector $\bar{b}$.

The original Byzantine Generals problem was introduced to model protocols that function correctly despite the failure of components of a computer system, where perhaps there is no external information that can be independently verified. In contrast, our modified variant of the Byzantine Generals problem can be thought of as a metaphor for scenarios (that are common these days) where there is a need to ascertain factual information that is spread across many sources, and the cost for an individual to validate every fact would be prohibitively expensive. This raises a need for a collaborative protocol to share these costs, where, as in the original Byzantine Generals Problem, there are untrustworthy peers who might communicate false findings.

Formally, our model represents the process of making observations as a trustworthy external "read-only" data source. Making observation i is modeled as sending a query on b_i to this data source. Note that in the DR model, making the right choice is *always possible* (albeit at a very high cost); each peer can learn each observation bit b_i by querying the source directly. Hence, a significant advantage of this approach is that impossibility concerns are eliminated; reaching an agreement is always possible, for any number of corrupt peers. Instead, the focus shifts to efficiency: the peers must collaborate to share the cost of learning the observation bits from the source.

The DR model consists of a peer-to-peer network and an external data source in the form of an n-bit array $\mathbf{X}$. There are k peers, some of which may be Byzantine. They can query information from the array and communicate with each other through messages. The peers initially do not know the array's contents, but are required to compute some function of the n bits. The peers can learn information about the bits either by querying the data source directly or from other peers. The primary goal is to minimize the maximum number of queries by any honest peer.

The DR model gives rise to a natural class of problems, called *retrieval problems*. In a retrieval problem, each peer needs to output $f(\mathbf{X})$ for some computable function f of the input. In this work, we focus on the specific problem where f is the *identity* function,

hereafter referred to as the Download problem. We view this as the most fundamental retrieval problem since every computable function f of the input can be computed by the peers by first running a Download protocol and then computing $f(\mathbf{X})$ locally at no additional query, communication, or time cost. Hence, the query cost of the Download problem serves as a baseline against which to compare the costs of other specialized protocols for specific problems. Observe that a lower bound on query complexity for computing any Boolean function on $\mathbf{X}$ serves as a lower bound for Download as well.

The DR model was introduced in [4] and was inspired by distributed oracle networks (DONs) which are a part of blockchain systems. In such networks, peers are tasked with retrieving information from external data sources such as stock prices via well-defined *Application Interfaces* (APIs). Calls to these APIs incur a cost that increases with the oracle's usage. Thus, reducing the number of API calls also reduces the overall cost of the DON.

The Download problem is easily solved in a query-balanced manner in the absence of failures. This problem becomes challenging in the presence of faults. We consider a synchronous[1] failure-prone setting where up to βk peers are Byzantine and at least $\gamma \geq 1-\beta$ fraction are honest. For such systems, a tight bound on query complexity is established in [4] for deterministic Download, complemented by two randomized protocols that solve Download w.h.p.[2] The first can tolerate any constant fraction $\beta < 1$ of Byzantine faults but has non-optimal query complexity of $O\left(\frac{n}{\gamma k} + \sqrt{n}\right)$, while the second has optimal query complexity of $\tilde{O}\left(\frac{n}{\gamma k}\right)$ but can only tolerate up to $\beta < \frac{1}{3}$ fraction of Byzantine faults[3]. In this paper, we explore both the query and time complexity of the Download problem, with our main contribution being a novel randomized synchronous protocol that combines the best of both results, achieving optimal query complexity while tolerating any fraction $\beta < 1$ of Byzantine faults.

1.1 The Model

The Data Retrieval model consists of **(i)** k peers that form a *clique* and **(ii)** a source of data that is external to the clique called the *source* that stores the n-bit input array and provides read-only access to its content through queries.

Clique network and communication mode. In a clique (complete) network, the k peers are identified by unique IDs, assumed to be from the range $[1, k]$. In each round, every peer can send a message of up to b bits to each of the other peers. The common variant of this communication mechanism, referred to as *peer-to-peer message passing* communication, allows a peer to send in each round a *different* message to each of the other peers. However, we also discuss (in Sect. 4.3) a variant termed *broadcast* communication, where each peer (including a Byzantine one) can send at most one message per round, and that message is delivered to all other peers. This variant can be implemented in a peer-to-peer network using Byzantine Reliable Broadcast, albeit at additional time and message costs. However, there are communication networks, such as radio networks, where the behavior of communication matches the description of our broadcast model.

[1] The companion paper [6] considers the *asynchronous* setting, which behaves rather differently.

[2] Throughout this paper, we use the term "with high probability" to mean the following: with probability $\geq 1 - n^{-c}$, for some constant $c \geq 1$. We often abbreviate the phrase "with high probability" as "whp" or "w.h.p."

[3] We use the $\tilde{O}(\cdot)$ notation to hide β factors and polylogarithms in n and k.

The source. The n-bit input array[4] $\mathbf{X} = \{b_1, \ldots, b_n\}$ is stored in the source. It allows peers to retrieve that data through queries of the form $\texttt{Query}(i)$, for $1 \leq i \leq n$. The answer returned by the source would then be b_i, the true i^{th} element in the array. This type of communication is referred to as *source-to-peer* communication. Note that we assume the source to be a completely passive entity, and thus it cannot perform computations (such as error correcting codes) as part of the execution of protocols.

Network delay and rounds. We consider a synchronous setting where peers share a global clock and time is partitioned into rounds as follows. Each round consists of three sub-rounds:

1. The *query sending sub-round*, in which each peer can send q queries ($0 \leq q \leq n$) of the form $\texttt{Query}(\cdot)$ to the source.
2. The *query response sub-round*, in which the source responds to all the queries.
3. The *message-passing sub-round* of peer-peer communication, which consists of message exchange between peers. Every message is of size $O(\log n)$ unless otherwise stated.

We assume that local computation is instantaneous and is performed at the beginning of each sub-round. We also assume that a peer can choose to ignore (not process) messages received from another peer during the execution. Such messages incur no communication cost to the recipient peer.

The adversarial settings. The behavior of the environment in which our protocols operate is modeled by an adversary $\mathcal{A}$ that selects the input data, and determines the peers' failure pattern, but it cannot corrupt the communication with the source. In executing a protocol, a *Byzantine* peer can deviate from the protocol arbitrarily (controlled by $\mathcal{A}$). The adversary $\mathcal{A}$ can corrupt at most βk peers for some given[5] $\beta \in [0, 1)$. A peer is considered *honest* if it obeys the protocol throughout the execution. Letting $\gamma = 1 - \beta$, there is (at least) a γ fraction of honest peers. We denote the set of Byzantine (respectively, honest) peers in the execution by $\mathcal{F}$. (resp., $\mathcal{H}$).

The Byzantine adversary can select peers to corrupt at the start of each round. We consider two types of Byzantine adversaries. Under the *fixed adversary*, a corrupted peer remains corrupted for the rest of the execution. The *dynamic* adversary can decide on the set of corrupt peers arbitrarily at the start of any round, or more explicitly, it can make a peer v Byzantine on one round and honest on the next. In both cases, the total number of corrupted peers at any given time is bounded by βk, and the adversary can adaptively decide which peers to corrupt.

As in [4], we assume a strong adversarial model. At the beginning of each round t, $\mathcal{A}$ has knowledge of all the local random bits generated up to round $t - 1$ and all peer-peer and source-peer communications up to round $t - 1$.

Definitions. The following complexity measures and parameters are used in our analysis.

Query Complexity ($\mathcal{Q}$): the maximum number of bits queried by an honest peer during the execution,

Round Complexity ($\mathcal{T}$): the number of rounds (or *time*) it takes for the protocol to terminate,

Message Complexity ($\mathcal{M}$): the total number of messages sent by honest peers during the execution,

Message_size (ϕ): the maximum number of bits sent in one message by an honest peer during the execution.

---

[4] Throughout this paper, we assume $n \geq k$. In typical applications, $n \gg k$.

[5] We do not assume β to be a fixed constant (unless mentioned otherwise).

Table 1 Our main results (with β treated as *any* positive constant < 1). Here, * indicates results that only hold in expectation and # indicates results that hold in worst-case.

Comparison of the Existing and Developed Results for Byzantine Fault					
Adversary	**Theorem**	**Resiliency**	**Query**	**Time**	**Message Size**
Fixed Byzantine	[4]	$\beta < 1$	$\tilde{O}(n/k + \sqrt{n})$	$O(n)$	$O(\log n)$
	[4]#	$\beta < 1/3$	$\tilde{O}(n/k)$	$O(n)$	$O(\log n)$
	Thm 5	$\beta < 1$	$\tilde{O}(n/k)$	$O(n \log k)$	1
Dynamic Byzantine	Thm 8	$\beta < 1$	$\tilde{O}(n/k + \sqrt{n})$	2	$\tilde{O}(n/k + \sqrt{n})$
	Thm 10	$\beta < 1$	$\tilde{O}(n/k)$*	$O(\log k)$	$O(n)$
Dynamic Byzantine and Broadcast	Cor 11	$\beta < 1$	$\tilde{O}(n/k)$*	$O(\log n)$	$O(n/k)$
	Thm 12	$\beta < 1$	$\tilde{O}(n/k)$#	$O(\log^2 n)$	$\tilde{O}(n/k)$

Given that queries to the source generally incur higher costs, we focus on minimizing the query complexity $\mathcal{Q}$. Note that our definition of $\mathcal{Q}$ (measuring the maximum cost per peer rather than the total cost) favors a fair and balanced load of queries across honest peers.

The Download problem. Consider a DR network with k peers, where at most βk can be Byzantine, and a source that stores a bit array $\mathbf{X} = [b_1, \ldots, b_n]$. The Download problem requires each peer to learn $\mathbf{X}$. Formally, each honest peer μ outputs a bit array res_μ, and it is required that, upon termination, $res_\mu[i] = b_i$ for every $i \in \{1, \cdots, n\}$ and $\mu \in \mathcal{H}$, where $\mathcal{H}$ is the set of honest peers.

To solve this problem in the absence of failures, all n bits need to be queried, and this workload can be shared evenly among k peers, giving $\mathcal{Q} = \Theta(n/k)$. The message complexity is $\mathcal{M} = \tilde{O}(nk)$, assuming small messages of size $\tilde{O}(1)$, and the round complexity is $\mathcal{T} = \tilde{O}(n/k)$ since $\Omega(n/k)$ bits need to be sent along each communication link when the workload is shared.

1.2 Related Work

Following the model introduced in [4], our work studies a new class of fault-tolerant problems that is heavily inspired by Blockchain oracles. Our focus is on the synchronous setting, but we also explore the asynchronous version, which is quite different, in our companion paper [6]. There are multiple classic *Byzantine Fault-Tolerant (BFT)* problems (e.g., agreement, broadcast, and state machine replication) that provide insight and inspiration when considering the Download problem, cf. [9, 14, 19, 23, 24]. We were also influenced by protocols developed for Oracle networks, such as OCR and DORA [10, 13]. Our model and algorithms have a similarity to some work on recommendations systems, in particular [7]. See Sect. 5 for a more detailed comparative discussion.

1.3 Our Contributions

We explore the Download problem under various adversarial and network models and present several deterministic and randomized protocols and a lower bound for Download. Here, we state only simplified bounds, in which the $\tilde{O}(\cdot)$ notation hides factors dependent on β and poly log factors in n. The main results are summarized in Table 1 for convenience.

Query Optimality with Synchronous Point-to-point Communication and Byzantine Failures. We start with closing the gap left open in [4]. The model studied in that paper involves a synchronous point-to-point communication network and Byzantine failures. In this model, for deterministic protocols, the Download problem turns out to be expensive, requiring $\Omega(\beta n)$

queries in the worst case (a matching upper bound is shown which works under asynchrony as well). Every peer essentially has to query the entire input array for itself. However, [4] gives a randomized protocol that solves the Download problem (and consequently *any* function of the input) for an *arbitrary* fraction $\beta < 1$ of Byzantine faults while requiring at most $\tilde{O}(n/k + \sqrt{n})$ queries per peer. The result is nearly as efficient as the *failure-free model* whenever $k < \sqrt{n}$. The time and message costs are $\mathcal{T} = O(n)$ and $\mathcal{M} = \tilde{O}(kn + k^2\sqrt{n})$. A natural question then, is whether the additive $\sqrt{n}$ term is necessary for $k > \sqrt{n}$. It was shown in [4] that as long as $\beta < 1/3$, one can be fully efficient for all $k \in [1, n]$, getting $\mathcal{Q} = \tilde{O}\left(\frac{n}{k}\right)$, $\mathcal{T} = \tilde{O}(n)$, and $\mathcal{M} = \tilde{O}(nk^2)$. In Section 2 we close the gap and show the existence of a randomized Download protocol with query complexity $\mathcal{Q} = O\left(\frac{n}{\gamma k}\right)$, $\mathcal{T} = O(n \log k)$, and $\mathcal{M} = O(nk^2)$ for $\beta \in [0, 1)$. In that protocol, we use a *blacklisting* technique, i.e., during an execution, honest peers can *blacklist* Byzantine ones, after identifying a deviation from the behavior expected of an honest peer, and subsequently ignore their messages. A Byzantine peer can be blacklisted for a variety of reasons, e.g., if it is directly "caught" in a lie about some bit value, or stops sending messages while they are expected of it, or sends more messages than it is expected to. The protocol is carefully constructed such that the amount of "noise" that can be caused by Byzantine peers, is bounded, and does not cause a large overhead for the honest peers.

Faster Query-optimal Solutions with Synchronous Communication and Byzantine failures. We next ask whether Download can be achieved faster than linear time. To exclude the extreme end of the scale, we show (in Sect. 3) that hoping for a single round Download (assuming arbitrarily large messages can be sent in a single round) is too ambitious. We show that every randomized protocol in which each peer queries no more than $(n - 1)$ bits fails to solve Download with constant probability greater than 0 (for $\beta \approx 1/2$).

Nevertheless, we next derive faster protocols (in Section 4) than the one in Section 2. Specifically, in Subsection 4.1 we show how Download can be achieved in *two* rounds. This solution enjoys the additional advantage that it can cope with a stronger type of adversary, termed *dynamic adversary*, which can change the set of Byzantine peers from one round to the other, provided that this set never exceeds a size of βk. To achieve this result, during the execution of the protocol, we consider a multi-set of the bit strings proposed by different peers that purport their respective strings to be equal to a particular interval of the input bit array. If it is known that at least $t > 0$ proposers were honest peers that correctly know the interval, we can discard all strings in the multi-set that do not appear at least t times, and conclude that the remaining set of distinct strings contains a correct string. A *decision tree* for a set S of strings is a rooted binary tree. Each internal node x is labeled by an index i of the input array $\mathbf{X}$ and each leaf is labeled by a string s such that if a root-to-leaf path goes to the left subtree of a node labeled i, then the i^{th} bit of s is 0, else it is 1. Given a set of strings S of which one is consistent with $\mathbf{X}$, one can build a decision tree with $|S| - 1$ nodes and determine the correct leaf with $|S| - 1$ simultaneous queries.

Unfortunately, this protocol no longer attains query-optimality; its query complexity is $O(n/(\gamma k) + \sqrt{n})$. We improve this query complexity in Subsection 4.2 where we describe an iterated version of the 2-step protocol with *expected* query complexity $O(n \log n/(\gamma k))$ and $O(\log n)$ time. Note that in some settings, large messages can be simulated by several rounds of smaller messages, but here, in a randomized setting with an adaptive adversary, it may not be possible. An adversary can observe the smaller messages that arrive piecemeal over multiple rounds and may be able to adaptively corrupt peers more effectively.

Finally, in Subsection 4.3 we show how the same nearly optimal results can be achieved, with worst case query complexity and small message size, provided we assume *broadcast* communication among peers rather than point-to-point. That is, in each round, each peer must send the same message to all other peers, and this applies also to the Byzantine peers. In this model, we get a protocol with the worst case query complexity $O((1/\gamma)\log^2 n)$ and $O((1/\gamma)\log^2 n)$ time, and message size $O(\log n/\gamma)$. A lower bound in Section D shows that this protocol demonstrates an optimal trade-off among these parameters, up to $\log n$ factors.

2 Query-optimal Download

In this section, we show how to optimize the query complexity of Download up to a factor of $\log n$, achieving $\mathcal{Q} = O\left(\frac{n \log n}{\gamma k}\right) = \tilde{O}\left(\frac{n}{\gamma k}\right)$ for any $\beta < 1$. This provides an improvement over the best previously known results for Download [4], which either guaranteed a query complexity of $\mathcal{Q} = \tilde{O}\left(\frac{n}{\gamma k} + \sqrt{n}\right)$ or imposed the additional restriction of $\beta < \frac{1}{3}$.

The protocol described next works when $2\gamma k > 2^{\delta} \cdot \lg^2 n$, for some constant δ[6]. Note that, for smaller values of k, the desired bound of $\tilde{O}\left(\frac{n}{\gamma k}\right)$ on the query complexity holds trivially: when $k \leq \left(\frac{2^{\delta}}{2\gamma}\right) \cdot \lg^2 n$, this bound is $\tilde{O}\left(\frac{n}{\gamma k}\right) = \tilde{O}(n)$, which is attainable by the trivial protocol where every peer queries the entire input vector.

Our protocol runs in n *epochs*. For $1 \leq i \leq n$, an honest peer μ dedicates the i^{th} epoch to learning the i^{th} input bit. Furthermore, each epoch i is divided into *phases*. During each phase j of the epoch i, μ attempts to learn the i^{th} input bit b_i in one of two ways:

(i) based on messages received from the other peers in earlier phases, or

(ii) by querying the source directly (which happens with gradually increasing probability).

μ relies on the messages of its peers for learning $b_i = b$ in phase j only if the number of peers who sent it the value b exceeds a specified threshold (depending on j) *and* the number of peers who sent it the value $(1 - b)$ is below that threshold. If μ fails to learn the bit in phase j, then it proceeds to phase $(j + 1)$, in which it doubles its probability of querying. This is repeated until μ successfully learns b_i. (Note that a head is always thrown when 2^j exceeds $\frac{\gamma k}{\lg n}$).

Once μ learns b_i, it broadcasts it and blacklists any peers that sent contradictory values. In subsequent phases, μ never relies on bits sent by blacklisted peers. See Algorithm 1 for the pseudocode. Note that in each epoch i, μ learns the value of the i^{th} bit either by *gossip learning* (i.e., receiving messages with a decisive majority, in line 9 of the protocol) or by *query learning* (in line 19). We name the epoch accordingly as a *gossip* epoch or as a *query* epoch. Furthermore, in every epoch i, μ performs a "learning step" in *exactly one* phase, hereafter referred to as the *learning phase* of epoch i and denoted $\ell(i)$.

Correctness. Let $P_j = 1 - \left(1 - \frac{1}{\gamma k}\right)^{2^j}$ be the probability that when flipping 2^j independent random coins, each with bias $\frac{1}{\gamma k}$ toward head, at least one turns heads. The following technical lemma provides upper and lower bounds on P_j.

▶ **Lemma 1.** *For $j \in [f, f^{end}]$ (see line number 6 in Algorithm 1), $P_j \in \left(\frac{2^j}{\gamma k}\left(1 - \frac{1}{2\lg n}\right), \frac{2^j}{\gamma k}\right)$.*

[6] For any positive real number x, we use the notation $\lg x$ to denote $\log_2 x$.

Proof. The binomial expansion of $(1-\varepsilon)^m$ yields the bounds $1-m\varepsilon < (1-\varepsilon)^m < 1-m\varepsilon+\frac{m(m-1)}{2}\varepsilon^2$. Setting $\varepsilon = 1/\gamma k$ and $m = 2^j$, we get $1-2^j/\gamma k < (1-1/\gamma k)^{2^j} < 1-2^j/\gamma k + 2^j(2^j-1)/(2(\gamma k)^2)$, or

$$\frac{2^j}{\gamma k} \;>\; P_j \;>\; \frac{2^j}{\gamma k} - \frac{2^j(2^j-1)}{2(\gamma k)^2} \;>\; \frac{2^j}{\gamma k}\left(1-\frac{1}{2}\frac{2^j}{\gamma k}\right) \;>\; \frac{2^j}{\gamma k}\left(1-\frac{1}{2\lg n}\right)\,,$$

where the last inequality follows from the assumption that $2^j < \gamma k/\lg n$. ◀

To prove the correctness of the protocol, we need to show that all honest peers μ compute $\mathbf{X}^\mu = \mathbf{X}$ correctly with high probability. This is shown by inductively proving that, for any honest peer μ and any bit i, $1 \le i \le n$, $b_i^\mu = b_i$ with high probability. The induction is on i for a fixed honest peer μ. We prove the following two invariant statements inductively.

Blacklisting Statements (BL_i). All peers blacklisted by some honest peer μ until the end of the i-th epoch (i.e., the i^{th} iteration of the outer `for` loop, line 2 of Algorithm 1) are indeed Byzantine, with high probability. For convenience, BL_0 refers to the empty blacklist before the execution enters the loop and is clearly true.

To capture the inner `for` loop (i.e., the phases) as well (see Algorithm 1, line 6), we use $BL_{i,j}$, for $j \in [f, f^{end}]$ and $1 \le i < n$, to refer to the statement that all peers blacklisted by some honest peer μ until the end of the j^{th} inner `for` loop of the $(i+1)$-th epoch are indeed Byzantine with high probability. For convenience, we may use $BL_{i,f-1}$ to refer to BL_i. Also, note that $BL_{i,f^{end}}$ implies BL_{i+1}.

Correctness Statement (C_i). All honest peers have executed the first i epochs correctly, with high probability, i.e., $b_{i'}^\mu = b_{i'}$ for all $1 \le i' \le i$ and all honest peers μ. Again, C_0 refers to the execution being vacuously correct before any bit is processed by the peers. Furthermore, $C_{i,j}$, $1 \le i < n$ and $j \in [f, f^{end}]$, refers to the following statement at the end of phase j in epoch $(i+1)$: for all honest peers μ, $b_{i'}^\mu = b_{i'}$ for all $1 \le i' \le i$ and $b_{i+1}^\mu = b_{i+1}$ for all honest peers that have set their respective I_{voted} bit to 1, with high probability. For convenience, we sometimes use $C_{i,f-1}$ to refer to C_i. Also, note that $C_{i,f^{end}}$ implies C_{i+1}.

Observe that peer μ performs a query if (a) it is the first phase of that particular epoch and it has obtained "heads", or (b) it is phase $j > f$, it is the first time μ has drawn "heads", and $COUNT_i^b \ge \nu 2^j$ for *both* $b = 0$ and $b = 1$.

The basis for the induction is provided by the statements BL_0 and C_0, which are both true. The inductive step is given by the following lemma.

▶ **Lemma 2.** *Consider some epoch $i \in [1, n]$ and assume that conditions BL_{i-1} and C_{i-1} hold at the start of epoch i. For any fixed $c \ge 1$, there is a suitably small fixed choice for the constant δ such that the statements BL_i and C_i hold with probability at least $1 - \frac{1}{n^{c+1}}$ at the end of the epoch i.*

Proof. Fix $c \ge 1$. To prove the claim, we show that $BL_{i-1,j}$ and $C_{i-1,j}$ hold for all phases $j \in [f, f^{end}]$. By definition, $BL_{i-1,f^{end}} = BL_i$ and $C_{i-1,f^{end}} = C_i$. For the sake of contradiction, suppose statement C_i does not hold, and let j^* be the first phase j in epoch i when statement $C_{i-1,j}$ is false, namely, some peer μ tosses heads and goes on to set its I_{voted} bit to 1, but incorrectly assigns $b_i^\mu = 1 - b_i$. Since j^* is the first such occurrence of incorrect behavior, BL_{i-1,j^*-1} and C_{i-1,j^*-1} are true. If $j^* = f$, then μ will explicitly query the bit, so $b_i^\mu = b_i$, a contradiction. So, we focus on the case where $j^* > f$.

Without loss of generality, let $b_i = 0$. We claim that the number of votes received by μ in favor of $b_i = 0$ will be at least $\nu \cdot 2^{j^*} = 2^{j^*-2}$ with probability at least $1 - \frac{1}{n^c}$. Since BL_{i-1,j^*-1} and C_{i-1,j^*-1} are true, all honest peers that tossed heads in earlier phases $j < j^*$ would have correctly voted for $b_i = 0$. To establish the required contradiction, we show that the number of such votes is at least $\nu \cdot 2^{j^*} = 2^{j^*-2}$. Let G_{old} be the set of votes received by μ from honest peers in epoch i during phases j for $j \leq j^* - 2$ (if such phases exist). Let G_{recent} be the set of votes received by μ from honest peers in phase $j' = j^* - 1$; we know this phase exists as $j^* > f$. Our goal is to show that $X = G_{old} \cup G_{recent}$ has cardinality $|X| \geq 2^{j^*-2}$ with high probability. Let G denote the set of all honest peers; $|G| \geq \gamma k$. Note that for every peer in $G \setminus G_{old}$, the probability of joining G_{recent} is $P_{j'} = P_{j^*-1}$ by Lemma 1, so the expected size of X is

$$\mathbb{E}[|X|] \;\geq\; |G| \cdot P_{j^*-1} \;\geq\; \gamma k \cdot \frac{2^{j^*-1}}{\gamma k}\left(1 - \frac{1}{2\lg n}\right) \;=\; 2^{j^*-1}\left(1 - \frac{1}{2\lg n}\right) \;\geq\; 0.9 \cdot 2^{j^*-1},$$

for sufficiently large n (say, $n \geq 32$). Thus, applying Chernoff bound, we get

$$\begin{aligned}
\Pr[|X| \;<\; 2^{j^*-2}] \;&=\; \Pr\left[|X| < 0.5 \cdot 2^{j^*-1}\right] \;\leq\; \Pr\left[|X| < (1 - 4/9)\mathbb{E}[|X|]\right] \\
&<\; e^{-\frac{(4/9)^2}{2}\mathbb{E}[|X|]} \leq\; e^{-\frac{(4/9)^2}{2}\cdot 0.9 \cdot 2^{j^*-1}} \leq\; e^{-\frac{4}{45}\cdot 2^f} \;=\; e^{-\frac{4}{45}\cdot 2^{\delta + \lg\lg n}} \;=\; e^{-\frac{4}{45}\cdot 2^{\delta}\cdot \lg n} \\
&<\; n^{-\frac{4}{45}\cdot 2^{\delta}} \;<\; \frac{1}{n^{c+2}},
\end{aligned}$$

where the last inequality holds when $\delta \geq \lg(45(c+2)) - 2$.

Thus, the number of votes received by μ for the correct bit value (0, as per our assumption wlog) must be at least $\nu \cdot 2^{j^*}$ relying on the fact that by the inductive assumption BL_{i-1,j^*-1}, no honest peer was added to B. From the else part (see line 15 of Algorithm 1), if both bit values received $\nu \cdot 2^j$ votes (or more), then μ will query bit i and this will ensure C_{i-1,j^*}. On the other hand, if the number of votes for bit value 1 is less than $\nu 2^{j^*}$, then, μ will vote for 0, which will be the correct bit. Thus, the contradiction is established, thereby implying C_i with probability at least $1 - \frac{1}{n^{c+1}}$ (applying the union bound over all peers μ).

If C_i holds, then all other honest peers $\mu' \neq \mu$ also vote correctly for 0, so μ will not blacklist any of them. This implies that the invariant BL_i also holds. ◀

Applying Lemma 2 and taking the union bound over all n epochs, we get the following.

▶ **Corollary 3.** *For δ fixed as in Lemma 2, Algorithm 1 ensures that all honest peers μ correctly compute $\boldsymbol{X}^\mu = \boldsymbol{X}$ with probability at least $1 - \frac{1}{n^c}$.*

Query complexity analysis. In the absence of Byzantine peers, the total number of (necessary) queries is $O(n)$, so the average cost per peer is $\mathcal{Q} = O\left(\frac{n}{\gamma k}\right)$. The reason for the additional wasteful queries is two-fold. First, the fact that the protocol is randomized and must succeed in learning all bits with high probability requires some redundancy in querying. Second, Byzantine peers spread fake information, forcing honest peers to perform queries to blacklist the culprits and clarify the true values of $\mathbf{X}$.

Let $\mathbf{R}(\mu)$ denote the sequence of independent random coins flipped by peer μ during execution (each with a bias of $\frac{1}{\gamma k}$ towards heads). For notational convenience, we will often write $\mathbf{R}$ to mean $\mathbf{R}(\mu)$, when the underlying peer μ is clear from the context. Let $\mathbf{R}_{i,j}$ denote the subsequence of $\mathbf{R}$ that was drawn at the beginning of phase j of the epoch i, and let $\mathbf{R}_i$ denote the subsequence of $\mathbf{R}$ that was drawn during the entire epoch i. Respectively, let $R = |\mathbf{R}|$, $R_i = |\mathbf{R}_i|$ and $R_{i,j} = |\mathbf{R}_{i,j}|$.

The variable $S_{i,j}$ used in the protocol denotes the number of coins of $\mathbf{R}_{i,j}$ that turned heads. Similarly, let S_i denote the number of coins of $\mathbf{R}_i$ that turned heads.

▶ **Lemma 4.** *For any fixed $c' \geq 1$, there is a suitably small fixed choice for the constant δ such that Algorithm 1 has query complexity $\mathcal{Q} = O\left(\frac{n \log n}{\gamma k}\right)$ with probability $1 - \frac{1}{n^{c'}}$.*

Proof. Throughout the proof, we consider an honest peer μ.

For every $1 \leq i \leq n$, let $B(i)$ denote the number of peers that were blacklisted by μ in epoch i and let $C(i)$ denote the number of queries performed by μ in epoch i. (Note that $C(i)$ is 0 if i is a *gossip epoch* and 1 if i is a *query epoch*.)

As $C(i) = 0$ for a gossip epoch i, the total cost of gossip epochs is

$$C^{gossip} \;=\; \sum_{\text{gossip } i} C^{gossip}(i) \;=\; 0. \tag{1}$$

Hence, we only need to analyze query epochs. We bound the number of queries in these epochs by first bounding the number of random coins flipped in these epochs.

Consider such an epoch i, with learning phase $\ell = \ell(i)$. There are two cases. The first is when $\ell = f$. In this case, perhaps no Byzantine peers were blacklisted, so $B(i) \geq 0$ and the number of random Coins flipped in this epoch is $R_i = R_{i,f} = 2^f$.

The second case is when $\ell > f$. The fact that μ had to query in phase ℓ implies that gossip learning was not possible, so both $COUNT_i^0 \geq \nu 2^j = 2^{\ell-2}$ and $COUNT_i^1 \geq 2^{\ell-2}$. This, in turn, implies that the number of Byzantine peers that μ gets to blacklist in this epoch is $B(i) \geq 2^{\ell-2}$. On the other hand, the number of Coins used during this epoch is $R_i = \sum_{j=f}^{\ell} R_{i,j} = 2^{\ell+1} - 2^f$.

Combining both cases, we get that for every epoch i, $B(i) \geq (R_i - 2^f)/8$. Summing over all i, we get

$$\beta k \;\geq\; B \;=\; \sum_i B(i) \;\geq\; \sum_i \frac{R_i - 2^f}{8} \;=\; \frac{R - 2^f n}{8}\,.$$

Rewriting, and recalling that $f = \lceil \delta + \lg \lg n \rceil$, we get

$$n \cdot 2^\delta \lg n \;\leq\; R \;\leq\; 8\beta k + 2^f n \;\leq\; 8\beta k + n \cdot 2^\delta \lg n, \tag{2}$$

where the first inequality follows from the fact that in every epoch i, $R_i \geq R_{i,f} \geq 2^\delta \lg n$.

Note that $C(i) \leq S_i$ for every i, hence

$$C^{query} \;=\; \sum_{\text{query } i} C(i) \leq \sum_{\text{query } i} S_i \;=\; S. \tag{3}$$

Recalling that $\mathbf{R}$ is a sequence of R independent Bernoulli variables with probability $\frac{1}{\gamma k}$ for heads, we have that $\mathbb{E}[S] = \frac{R}{\gamma k}$, and applying Chernoff bound we get

$$\Pr[S > 2R/\gamma k] \;=\; \Pr[S > 2\mathbb{E}[S]] \;\leq\; e^{-\mathbb{E}[S]/3} \;=\; e^{-R/3\gamma k} \;\leq\; e^{-n2^\delta \lg n/3\gamma k} \;\leq\; \frac{1}{n^{c'}}\,, \tag{4}$$

where the penultimate inequality relies on the left side of Eq. (2) and the last one holds when $\delta \geq \lg(3\gamma c')$, also relying on the assumption that $k \leq n$. Combining Equations (3), (4), and the right side of (2), we get that with probability at least $1 - \frac{1}{n^{c'}}$,

$$C^{query} \;\leq\; S \;\leq\; \frac{2R}{\gamma k} \;\leq\; \frac{2(8\beta k + cn \lg n)}{\gamma k} \;=\; O\left(\frac{n \log n}{\gamma k}\right). \tag{5}$$

Finally, the lemma follows by Equations (1) and (5). ◀

Each epoch lasts $O(\log(\gamma k))$ phases, so the time complexity is $O(n \log(\gamma k))$. Each peer sends at most n bits to each of the other machines, so the total number of messages sent is $O(nk^2)$. Moreover, the messages sent in line numbers 10 and 20 of Algorithm 1 can be encoded as one-bit messages as μ and i are known from context. Thus, we have the following theorem.

▶ **Theorem 5.** *In the synchronous point-to-point model with Byzantine failures, there is a randomized protocol for* Download *such that, with high probability,* $\mathcal{Q} = O\left(\frac{n \log n}{\gamma k}\right)$, $\mathcal{T} = O(n \log(\gamma k))$, *and* $\mathcal{M} = O(nk^2)$. *Moreover, it only requires single bit messages.*

3 Lower Bounds

Having established the existence of a query-optimal protocol for Download with an arbitrary bound $\beta < 1$ on the fraction of Byzantine peers, we turn to the issue of time complexity and derive faster Download protocols. In this section, we first establish that any *single*-round protocol is inefficient, requiring $\mathcal{Q} = n$. Then, we utilize information theory to show a lower bound on the trade-off between query complexity, time complexity and messages size.

Indistinguishability arguments typically require showing that two carefully constructed executions with different outputs "look the same" from the point of view of at least one peer μ that is honest in both of them. Hence, the peer μ will behave the same (and will output the same value) in both executions. In the well-studied Byzantine agreement context, the prover must show that a peer has the same input and will receive the same messages in both executions. In the case of randomized protocols, this becomes more complicated. Even under a fixed adversarial strategy, there is a probability space of executions over the random choices of all the peers. Therefore, we carefully construct probabilistic arguments over these execution spaces to show indistinguishability. In the DR model, these arguments are further complicated by the additional element of queries. Even if we show that the input of an honest peer and the messages it received are the same in both executions if the queries made by this peer in the different executions are not the same, then those executions do not "look the same" from its point of view and we can't say that it will act the same in the different executions. We obtain the following lower bound.

▶ **Theorem 6.** *Consider an n peer system with external data, for odd n, and let the fraction of Byzantine peers be bounded above by $\beta = (n-1)/2n$. For every protocol $A \in \mathcal{A}(n-1, 1)$, there is an adversarial strategy such that A fails to learn all the input bits with probability at least $(1 - e^{-1})/4$. The lower bound holds even in the broadcast communication model.*

Due to space considerations, readers interested in the proof details are referred to the full version of the paper [5].

4 Faster Download with Near Optimal Query Cost

In this section, we show how to reduce the time complexity to $O(\log n)$. Unlike the previous protocol, the protocols here do not use blacklisting and do not require fixed IDs from round to round; only that there is a known lower bound on the number of honest peers present in each round.

We first describe a key definition and a subroutine in these protocols. In a typical step of these protocols, we fix a parameter φ depending on the round, and the input vector $\mathbf{X}$ is partitioned into $\mathcal{K} = \lceil n/\varphi \rceil$ contiguous subsets of size φ, $\mathbf{X}[\ell, \varphi] = (b_{(\ell-1)\cdot\varphi+1}, b_{(\ell-1)\cdot\varphi+2}, \ldots, b_{\ell\cdot\varphi})$, for $\ell \in [1, \mathcal{K}]$, with the last interval, $(b_{(\mathcal{K}-1)\cdot\varphi+1}, \ldots, b_n)$), being possibly shorter. Each honest

peer queries all the bits of some interval $\mathbf{X}[\ell, \varphi]$ and broadcasts its findings to all other peers, in the form of a pair $\langle \ell, s \rangle$, where s is the bit string obtained from the queries. Hence at the end of a round, every peer receives a collection of strings for different intervals. Let $s[j]$ denotes the j-th bit of the string s. Note that a message $\langle \ell, s \rangle$ can possibly be received in multiple copies, from different peers, and the protocol keeps all copies.

Let S be a multiset of strings for a particular interval ℓ received by peer μ at the current round. Define the set of *t-frequent* strings at a peer μ as

$$\mathsf{FS}(S, t) = \{s \in S \mid s \text{ appears } \geq t \text{ times in } S\}$$

Note that the FS function gets a *multiset* and returns a *set* of t-frequent strings.

A *decision-tree* T is a rooted ordered binary tree where leaves are labeled with strings and each internal node v is labeled by an index i of the input array such that all the strings s in the left (respectively, right) sub-tree of v has $s[i] = 0$ (resp., $s[i] = 1$). Given a set of strings FS of which one is consistent with the input array (i.e., it appears exactly in the input array at the specified indices), we can build a decision tree to determine which one of the strings in FS is consistent with the input array with a cost of $|\mathsf{FS}| - 1$ queries, by querying the indices associated with all the internal nodes of the decision tree at the same time. The path consistent with the results of these queries reaches a leaf that is labeled by the correct string for the interval. See Algorithm 2.

4.1 A Simple 2-step Protocol

To illustrate the ideas of the following protocols in this section, we first give a 2-step protocol with $O(n/\sqrt{\gamma k})$ queries which succeeds with high probability. Note that if γk is very small, every honest peer queries every bit and the protocol is always correct. Otherwise:

1. Split the n bit string into $\mathcal{K}$ equal length intervals of φ bits each. Each peer picks an interval uniformly at random, queries all its bits, and broadcasts the discovered string s together with the identifier $\ell \in [1, \mathcal{K}]$ of its chosen interval.
2. Let FS_ℓ be the set of strings, each of which was received in at least $t = \gamma k/(2\mathcal{K})$ messages from distinct honest peers for interval ℓ, disregarding any string sent by a peer which sends more than one string. Formally, $\mathsf{FS}_\ell = \mathsf{FS}(S_\ell, t)$ where $S_\ell = \{s \mid \langle \ell, s \rangle \text{ received from some peer}\}$.
   In parallel, for each interval $\ell \in [1, \mathcal{K}]$, build a decision tree T_ℓ from FS_ℓ and invoke Procedure DETERMINE$(\cdot)$, letting each peer determine the correct string for interval ℓ.

Correctness. Consider an execution of the protocol, and denote the number of honest peers that pick the interval ℓ by k_ℓ. The protocol succeeds if $k_\ell \geq t$ for every interval ℓ, since every decision tree T_ℓ will contain a leaf with the correct string for the interval ℓ returned by Procedure DETERMINE.

▷ Claim 7. For constant $c \geq 1$, if $t \geq 8(c+1)\ln n$, then $k_\ell \geq t$ for every interval $\ell \in [1, \mathcal{K}]$, with probability at least $1 - 1/n^c$.

Proof. Fix $\ell \in [1, \mathcal{K}]$. The expected number of honest peers that pick the interval ℓ is $\mathbb{E}[k_\ell] = \gamma k/\mathcal{K} = 2t$. Therefore, applying Chernoff bounds, $\Pr[k_\ell < t] \leq e^{-t/8} \leq 1/n^{c+1}$ for $t \geq 8(c+1)\ln n$. Taking a union bound over all intervals, the probability that $k_\ell < t$ for *any* interval ℓ is less than $1/n^c$. ◁

Query complexity. The analysis of the query complexity is based on the fact that the query cost in Step 2 for interval ℓ is the number of internal nodes of the decision tree, which is $|\mathsf{FS}_\ell| - 1$. We defer the full analysis to Appendix B

Message size: The message size in this protocol is the number of bits required to describe the first index of the interval chosen and its bits in Step 1, i.e., $O(\varphi + \log n) = O(\varphi) = O(\sqrt{n/\gamma} + n \log n/(\gamma k))$. In other words, the message size is not more than the query complexity in the first round which is not more than $\mathcal{Q}$, i.e., $O(\mathcal{Q})$.

▶ **Theorem 8.** *There is a 2-round randomized protocol for* Download *in the point-to-point model with* $\mathcal{Q} = O(n \log n/(\gamma k) + \sqrt{n/\gamma})$ *and message size* $O(n \log n/(\gamma k) + \sqrt{n/\gamma})$.

4.2 Download with $O(n \log n/(\gamma k))$ Expected Queries and $O(\log \gamma k)$ time

The 2-step protocol presented in the previous subsection, while fast and simple, is not query-optimal, with its main source of overhead being that every peer, after its initial query of the randomly selected interval, builds a decision tree for every other interval and determines the correct leaf (and hence the correct string) by performing additional queries. In this subsection, we extend the previous protocol, achieving optimal query complexity at the cost of going from $O(1)$ to $O(\log n)$ rounds.

The protocol is based on the following idea. Let $\varphi = \lceil (n/(\gamma k))(8(c+1)\ln n) \rceil$. In Step i, for $i \in [0, \ldots, \lceil \lg(n/\varphi) \rceil]$, the n bits are partitioned into $\mathcal{K}_i = \lceil n/\varphi_i \rceil$ intervals of size $\varphi_i = 2^i \varphi$. Each interval in step i consists of the concatenation of two consecutive intervals from step $i-1$, beginning with the first two intervals. In each step, each honest peer determines a random interval of increasing size by splitting it into its two interval parts and using the technique described in the 2-step protocol above on both parts, until the entire input vector is determined. This significantly reduces the number of decision trees constructed by each peer, from $\mathcal{K}$ to $O(\log n)$, which allows the improved query complexity.

See Algorithm 4 for a formal description.

Correctness. We defer some of the correctness proof to Appendix C, which is a straightforward extension of the argument in Section 4.1.

Denote by ℓ_μ^i the interval ID picked by peer μ at step i.

▶ **Lemma 9.** *In every step* $i \in [0, \lceil \lg(n/\varphi) \rceil]$*, every honest peer* μ *learns the correct value of* $\boldsymbol{X}[\ell_\mu^i, \varphi_i]$ *w.h.p.*

The correctness of the protocol follows from observing that at the last round $\ell = \lg(n/\varphi)$, the intervals are of size $\varphi_\ell = 2^\ell \varphi = n$. Hence, by Lemma 9, every peer learns the entire input.

Query complexity. Given an interval ℓ in Step $i+1$, let x be the total number of strings received for the subintervals in Step i that compose interval, i.e., $x = x_L + x_R$, where x_u for $u \in \{L, R\}$ denotes the number of strings received for subinterval ℓ_u. Let m_x be the number of intervals in Step $i+1$ such that the total number of strings received for that interval equals x. Formally,

$$m_x = |\{\ell \mid \text{received exactly } x \text{ messages of the form } \langle \ell_L, s, i \rangle \text{ or } \langle \ell_R, s, i \rangle\}|$$

The probability of picking an interval in Step $i+1$ with x strings is m_x divided by the total number of intervals, or $m_x/\mathcal{K}_{i+1}$. The expected cost of querying in Step $i+1$ is therefore

$$\sum_x \frac{m_x}{\mathcal{K}_{i+1}} \cdot \frac{x}{t_i} \;=\; \sum_x \frac{m_x}{\mathcal{K}_{i+1}} \cdot \frac{2x}{\gamma k/\mathcal{K}_i} \;=\; \frac{2\mathcal{K}_i}{\gamma k \cdot \mathcal{K}_{i+1}} \sum_x m_x \cdot x \;\le\; \frac{4}{\gamma}\,,$$

where the last inequality follows since each peer broadcasts at most one string, so $\sum_x m_x \cdot x \le k$.

Step 0 requires $\varphi = O(n \log n/(\gamma k))$ queries. The expected cost of querying is $O(1/\gamma)$ per step $i > 0$ per peer, and there are fewer than $\log n$ steps, so the total expected query cost is at most $O(n \log n/(\gamma k))$.

Message size. The maximum message size is that of the last round or $n/2 + 1$.

▶ **Theorem 10.** *There is a* $O\left(\log\left(\frac{\gamma k}{\log n}\right)\right)$*-round protocol which w.h.p. computes* Download *in the point-to-point model with expected query complexity* $\mathcal{Q} = O(n \log n/(\gamma k))$ *and message size* $O(n)$*.*

4.3 Broadcast Model: Worst-case $O((1/\gamma)\log^2 n)$ Queries and Time, and Message Size $O(\log n/\gamma)$

Here we start by showing that in the Broadcast model, where every peer (including those controlled by the adversary) must send the same message to all peers in every round, one can drastically reduce the message size in Algorithm 4. Next, we show that the bound on expected query complexity can be improved to a bound on the worst case query complexity because of the "common knowledge" property guaranteed to the peers by the broadcast medium. Specifically, our main results for the broadcast model are the following.

▶ **Lemma 11.** *There is a* $O(\log n)$*-round protocol that w.h.p. performs* Download *in the Broadcast model with expected query complexity and message size* $O(n \log n/(\gamma k))$*.*

▶ **Theorem 12.** *In the Broadcast model, there is a protocol with worst case* $O(\log^2 n)$ *time,* $O((n/\gamma k)\log^2 n)$ *queries, and* $O((n/\gamma k)\log n)$ *message size.*

Again, readers interested in the details are referred to the full paper [5].

5 Discussion of Related Work

Byzantine fault tolerance was first introduced by Pease, Shostak, and Lamport [23, 19] in 1980 and has since played a pivotal role in distributed computing. Throughout the 80's, the focus was on Byzantine Agreement [9, 19, 23, 24] along with some closely related problems like Byzantine Reliable Broadcast (BRB) [1, 14] and Collective Coin Tossing [11, 20]. Subsequently, Lamport introduced state machine replication (SMR) [17], which was extended to Byzantine resilience by Castro and Liskov [12]. SMR lies at the heart of distributed ledgers and blockchains, technologies that have gained popularity in the recent years.

In Byzantine Agreement, depending on the model assumptions and parameters, the Byzantine adversary can use various strategies to misrepresent or suppress critical information and foil agreement. In fact, even if communication is via broadcast, when $\beta \ge 1/2$, half of the input bits can be misrepresented, paving the way for a simple indistinguishability argument that shows that Byzantine Agreement is impossible. When communication is point-to-point

(and cryptographic primitives are not used), the Byzantine peers can equivocate, i.e., send inconsistent messages. Exploiting equivocation, a more subtle argument (see Chapter 5 in [3]) can be used to prove that Byzantine Agreement is impossible when $\beta \geq 1/3$.

A natural question then is to ask whether there are reasonable circumstances under which any fixed $\beta < 1$ can be tolerated. The answer is affirmative in a version of the Byzantine Resilient Broadcast [14] that allows peers to digitally sign messages. In the BRB problem, we have a designated leader, known to all peers, that has a message. All other peers must learn that message. The safety requirement of BRB, mandates that every honest peer outputs the same message, while the validity requirement demands that all honest peers must output the leader's message when the leader is honest. This problem is challenging because the leader may be Byzantine and can try to trick peers into having different views of the input message. Without using cryptographic primitives, BRB can only be solved when $\beta < 1/3$. However, BRB can be solved for any $\beta < 1$ when peers can digitally sign messages, a well-known result since the early days [14]. Intuitively, BRB with digital signatures makes it difficult for the Byzantine peers to suppress information or equivocate.

The Download problem first introduced in [4] also admits algorithms for any $\beta < 1$. The source is accessible to all nodes in the network, which makes it difficult for Byzantine nodes to suppress crucial information or equivocate without being caught. It is worth noting, however, that the Download problem is fundamentally different from BRB and other traditional problems. Unlike those problems, the Download problem admits a trivial protocol, tolerating any $\beta < 1$ even without the use of cryptographic primitives. In fact, each peer can directly query all the bits, but spending that many queries is prohibitively wasteful. Moreover, BRB protocols require active engagement from the leader, for signing messages, applying error-correcting codes and collision resistant hash functions, etc. In contrast, the source in the DR model is a read-only data source with no computational power, and therefore cannot be expected to execute such primitives. Thus, solutions for traditional problems such as BRB do not translate to solutions for Download. Importantly, with impossibility a non-issue for Download, our focus shifts to ensuring optimal query complexity (and other complexities as well) without sacrificing fault tolerance. This work shows that one can achieve both the optimal query complexity (within log factors) and resilience of up to $\beta < 1$ simultaneously. It is also worth noting that while using primitives like BRB for solving Download is possible, our solutions are significantly more efficient.

Recommendation Systems. The paper [7] studies the problem of determining the *preferences* (represented as Boolean values) of each peer for each of n objects. In that paper, in a single round, peers probe objects at unit cost per object and then post their results with their identity on a public billboard. For each peer, there is a presumed lower bound on the number of peers that have the same preferences, and this fact may be used, like in our algorithms, to determine preferences for objects that have not been probed. The model assumed in [7] is stronger than ours, with (1) a joint billboard, which facilitates broadcast (one-to-all) communication among peers, (2) global randomness, and (3) a static adversary that fixes its strategy before the start of the execution and cannot dynamically change it according to the random coins generated by the algorithm. In contrast, our model generally assumes point-to-point communication, uses only private randomness, and allows adaptive Byzantine behavior. Subsequent papers on recommendation systems consider variants such as lowering the cost by allowing the output to approximate the real preferences [2, 15, 21, 22], or considering the problem in a limited asynchronous environment where message delays (and order of probes) are fixed before the start of the execution [8]. These follow-up works use the same stronger model mentioned above.

Oracle networks. As mentioned above, in Oracle networks, a set of peers is assigned the task of bringing external off-chain data to the network, where a subset of these peers can be Byzantine. Generally, one can describe the operation of an Oracle network as follows. The network generates a report containing the observations made by some (sufficiently many) peers. Once a report is successfully generated, one or several peers transmit the report to an intermediary program (known as a smart contract) that runs on the blockchain, verifies the validity of the report, derives a final value from it (e.g., the median), and then exposes the value for consumption on-chain. Since the traditional usage of these networks is to track exchange rates (e.g USD-ETH) that change with time, studies on Oracle networks focus on the problem of creating a report where the derived value (say the median) must be acceptable (in the sense that it does not deviate much from the set of honest observations) while keeping the costs (e.g., of sending reports to the smart contract) low and tolerating as many Byzantine peers as possible, even at the expense of higher communication and computation off-chain. The Off-Chain Reporting (OCR) protocol [10] solves this problem with $\beta < 1/3$ by running a BA protocol to agree on $2f + 1$ values, then a designated *leader* generates a report and sends it to the contract (the leader acts as an aggregator). The Distributed Oracle Agreement (DORA) protocol [13] takes it a step further by using an approximate agreement scheme, the inherent ability of a Blockchain to act as an ordering service, and multiple aggregators. They improve results to sustain $\beta < 1/2$ and $\beta < 1/3$ w.h.p when the size of the Oracle network is significantly smaller than the size of the entire system. In both of these works, every peer reads all the external data and goes on to participate in the report generation. Our work complements the OCR and DORA protocols and focuses on how to efficiently read the off-chain data while minimizing the number of bits read per peer (as reading from an external source is also more costly than off-chain communication). Our approach would drastically reduce cost when the Oracle network keeps track of a large number of (static) variables (e.g., financial information). It could be used as a black box in the OCR and DORA protocols, and save the need to query all of the n values individually by each peer. Note that both OCR and DORA use cryptographic primitives, whereas we do not. A more in-depth description of the relation between the Download problem and Blockchain oracles is provided in the companion paper [6].

6 Conclusions and Future Work

In this work, we studied the Data Retrieval model, introduced in [4], and improved the previous results on the Download problem by achieving both optimal query complexity and optimal resiliency of any fraction $\beta < 1$ of Byzantine peers. In addition, we presented several new results, including a protocol with $O(\log n)$ time complexity and near-optimal expected query complexity in a model with a dynamic adaptive adversary, and, in the Broadcast model, near-optimal worst case time, query, and message complexity. We also established a lower bound for single-round protocols, demonstrating that it is necessary to query every bit to solve the Download problem within a single round.

Our work is useful in the context of blockchain oracles, specifically, as a sub-routine for data extraction from multiple data sources, where the data sources have some (possibly probabilistic) guarantee of being trustworthy. A relevant question in this context involves handling data that changes over time, such as stock prices or exchange rates. We leave this question open for future exploration, as these temporal changes must be formalized carefully and may require adjustments to the model.

References

1 N. Alhaddad, S. Das, S. Duan, L. Ren, M. Varia, Z. Xiang, and H. Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Proc. ACM Symp. on Principles of Distributed Computing*, pages 399–417, 2022.

2 N. Alon, B. Awerbuch, Y. Azar, and B. Patt-Shamir. Tell me who I am: An interactive recommendation system. *Theory Comput. Syst.*, 45:261–279, 2009. doi:10.1007/S00224-008-9100-7.

3 H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. Wiley, 2004.

4 J. Augustine, J. Biju, S. Meir, D. Peleg, S. Ramachandran, and A. Thiruvengadam. Byzantine Resilient Distributed Computing on External Data. In *38th Symp. on Distributed Computing (DISC)*, pages 3:1–3:23, 2024.

5 J. Augustine, S. Chatterjee, V. King, M. Kumar, S. Meir, and D. Peleg. Distributed download from an external data source in faulty majority settings. *CoRR*, abs/2412.19649, 2024. doi:10.48550/arXiv.2412.19649.

6 J. Augustine, S. Chatterjee, V. King, M. Kumar, S. Meir, and D. Peleg. Brief Announcement: Distributed Download from an External Data Source in Asynchronous faulty settings. In *DISC*, 2025. These proceedings.

7 B. Awerbuch, Y. Azar, Z. Lotker, B. Patt-Shamir, and M. R. Tuttle. Collaborate with strangers to find own preferences. *Theory Comput. Syst.*, 42:27–41, 2008. doi:10.1007/S00224-007-9016-7.

8 B. Awerbuch, A. Nisgav, and B. Patt-Shamir. Asynchronous active recommendation systems. In *Proc. 11th Conf. on Principles of Distributed Systems OPODIS 2007*, pages 48–61, 2007.

9 G. Bracha. Asynchronous byzantine agreement protocols. *Inf. & Computat.*, 75:130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

10 L. Breidenbach, C. Cachin, A. Coventry, A. Juels, and A. Miller. Chainlink off-chain reporting protocol. Technical report, Chainlink Labs, 2021.

11 A.Z. Broder and D. Dolev. Flipping coins in many pockets (byzantine agreement on uniformly random values). In *25th Symp. on Foundations of Computer Science*, pages 157–170, 1984.

12 M. Castro and B. Liskov. Practical byzantine fault tolerance. In *3rd Symp. on Operating Systems Design and Implementation*, OSDI, pages 173–186. USENIX Assoc., 1999.

13 P. Chakka, S. Joshi, A. Kate, J. Tobkin, and D. Yang. DORA: distributed oracle agreement with simple majority. *CoRR*, abs/2305.03903, 2023. doi:10.48550/arXiv.2305.03903.

14 D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

15 Seth Gilbert, Rachid Guerraoui, Faezeh Malakouti Rad, and Morteza Zadimoghaddam. Collaborative scoring with dishonest participants. In *Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 41–49, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1810479.1810488.

16 R. Impagliazzo and R. Williams. Communication complexity with synchronized clocks. In *25th IEEE Conf. on Computational Complexity*, pages 259–269, 2010.

17 L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133–169, 1998. doi:10.1145/279227.279229.

18 L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982. doi:10.1145/357172.357176.

19 L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, 1982. doi:10.1145/357172.357176.

20 S. Micali and T. Rabin. Collective coin tossing without assumptions nor broadcasting. In *CRYPTO*, pages 253–266, 1991.

21 A. Nisgav and B. Patt-Shamir. Finding similar users in social networks. *Theor. Comp. Sys.*, 49:720–737, 2011. doi:10.1007/S00224-010-9307-2.

22 A. Nisgav and B. Patt-Shamir. Improved collaborative filtering. In *22nd ISAAC*, pages 425–434, 2011.

23 M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980. doi:10.1145/322186.322188.

24 M. O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409, 1983.

A Supplementary Pseudocode

Algorithm 1 Resilient Download for any fixed known $\beta < 1$; code for peer μ.

$B \leftarrow \emptyset$ ▷ *Set of blacklisted peers*
for i from 1 to n **do** ▷ *Epoch i deals with input bit b_i*
  $I_{voted} \leftarrow 0$ ▷ *Indicator of learning the i-th bit and voting on it*
  $COUNT_i^0 \leftarrow 0$; $COUNT_i^1 \leftarrow 0$; ▷ *Counters for the number of peers that voted 0/1 for bit i*
  $f \leftarrow \lceil \delta + \lg\lg n \rceil$; $f^{end} \leftarrow \lceil \lg(\gamma k) - \lg\lg n \rceil$; ▷ *See Lem. 2 & 4 for choice of constant δ.*
  for j from f to f^{end} **do** ▷ *Phases within each epoch.*
    if $I_{voted} = 0$ (μ has not voted for bit i yet) **then**
      if $COUNT_i^{1-b} < \nu \cdot 2^j$ for $\nu = 1/4$ and for some $b \in \{0,1\}$ AND $(j \neq f)$ **then**
        $b_i^\mu \leftarrow b$ ▷ *Perform a "gossip learning" step*
        Broadcast a vote $\langle \mu, i, b_i^\mu = b \rangle$
        Set $I_{voted} \leftarrow 1$
      else
        if $j = f^{end}$ **then**
          Set coinflip to heads
        else
          Toss 2^j independent Coins with bias $1/\gamma k$ towards heads.
          Let $S_{i,j}$ be the number of Coins that turned heads. Set coinflip to heads if $S_{i,j} \geq 1$.
        if heads **then** ▷ *Perform a "query learning" step*
          $b_i^\mu \leftarrow b \leftarrow query(i)$; ▷ *$\mu$ learns $b_i^\mu = b$ by query*
          Broadcast a vote $\langle \mu, i, b_i^\mu = b \rangle$ ▷ *In practice, sending one bit suffices.*
          Set $I_{voted} \leftarrow 1$
      Receive messages from other peers.
      for $b \in \{0,1\}$ **do**
        $COUNT_i^b \leftarrow |\{\mu' \notin B \mid \mu$ received message $\langle \mu', i, b_i^{\mu'} = b \rangle$ during epoch $i\}|$
          ▷ *count voters for $b \in \{0,1\}$; ignore peers blacklisted in earlier epochs*
    if $I_{voted} = 1$ **then** ▷ *Including the case when I_{voted} changed during the current phase*
      $B \leftarrow B \cup \{\mu' \mid \mu'$ sent message with $b_i^{\mu'} \neq b_i^\mu\}$ ▷ *peers with contradictory vote*

Algorithm 2 CONST_DECISION_TREE(S)
input: Set of strings S
output: Node labeled tree T.

Create a root node v
if $|S| > 1$ **then**
  Set $i \leftarrow$ smallest index of bit on which at least two strings in S differ
  $label(v) \leftarrow i$
  Set $S^b \leftarrow \{s \in S \mid s[i] = b\}$ for $b \in \{0, 1\}$
  $T^0 \leftarrow$ CONST_DECISION_TREE(S^0)
  $T^1 \leftarrow$ CONST_DECISION_TREE(S^1)
  Let T be tree rooted at v with left-child(v) $\leftarrow T^0$ and right-child(v) $\leftarrow T^1$
else
  $label(v) \leftarrow s$, where $S = \{s\}$ ▷ $|S| = 1$, v *is a leaf*
  Let T be a tree consisting of the singleton v.
Return T

procedure DETERMINE(T)
  $J = \{j \mid \exists u \in T$ s.t. $j = label(u)\}$
  for all $j \in J$ **do** in parallel
    $query(j)$
  Let v be the root of T
  while v is not a leaf **do**
    Let $j = label(v)$.
    if $b_j = 0$ **then**
      Set $v \leftarrow$ left-child(v)
    else
      Set $v \leftarrow$ right-child(v)
  Return $label(v)$

Algorithm 3 2-Round Download with $\beta < 1$; Code for peer μ.

if $k \leq 32(c+1)\ln n/\gamma$ **then** query every bit and return
if $k < \sqrt{n/\gamma}\ln n$ **then** $\varphi \leftarrow \lceil 32(c+1)n\ln n/(\gamma k)\rceil$
else $\varphi \leftarrow \lceil 32(c+1)\sqrt{n/\gamma}\rceil$
$t \leftarrow \gamma k/(2\mathcal{K})$
Randomly select $\ell^\mu \in [1, \mathcal{K}]$
Set string $s^\mu \leftarrow query(\mathbf{X}[\ell^\mu, \varphi])$
Broadcast $\langle \ell^\mu, s^\mu \rangle$
for $\ell = 1$ to $\mathcal{K}$ **do** in parallel
  Construct the multiset $S_\ell \leftarrow \{s \mid \langle \ell, s\rangle$ received$\}$
  $T_\ell \leftarrow$ CONST_DECISION_TREE(FS(S_ℓ, t))
  $s^\ell \leftarrow$ DETERMINE(T_ℓ)
Output $s^1 s^2 \cdots s^{\mathcal{K}}$

B Query complexity analysis of Algorithm 3

The cost of querying in Step 2 for interval ℓ is the number of internal nodes of the decision tree, which is $|\mathsf{FS}_\ell| - 1$. Let x_ℓ be the number of strings received for interval ℓ in Step 1 (including copies). Then $|\mathsf{FS}_\ell| \leq x_\ell/t$. Since each peer sends no more than one string overall, $\sum x_\ell = k$. Hence, the cost of determining all intervals is $\sum_\ell |\mathsf{FS}_\ell| \leq \sum_\ell x_\ell/t \leq k/t$.

Algorithm 4 Download Protocol with $\tilde{O}(n/\gamma k)$ Expected Queries and $O(\log \gamma k)$ Time $\beta < 1$.

for $i = 0$ to $\lceil \lg(n/\varphi) \rceil$ **do**
  Randomly pick an interval $\ell \in [1, \mathcal{K}_i]$ of size φ_i
  if $i = 0$ **then**
    Set string $s = query(\mathbf{X}[\ell, \varphi])$ and broadcast $\langle \ell, s, 0 \rangle$
  else
    $\ell_L \leftarrow 2\ell - 1; \ell_R \leftarrow 2\ell$
    for $u \in \{\ell_L, \ell_R\}$ **do** in parallel
      Construct the multiset $S(u) \leftarrow \{s \mid$ received a message of the form $\langle u, s, i-1 \rangle\}$
      $t_{i-1} = 2^{i-2}\varphi(\gamma k/n)$
      $T_u \leftarrow$ Const_Decision_Tree($\mathsf{FS}(S(u), t_{i-1})$)
      $s_u \leftarrow$ Determine(T_u)
    Set s_ℓ to $s_{\ell_L} s_{\ell_R}$ and broadcast $\langle \ell, s_\ell, i \rangle$
return the determined string for interval $[1, \ldots, n]$

The query cost per peer, $\mathcal{Q}$, is the cost of determining every interval using decision trees plus the initial query cost; hence $\mathcal{Q} \leq k/t + \varphi$. Since $t = \gamma k/(2\mathcal{K})$, $\mathcal{Q} \leq 2\lceil n/\varphi \rceil (1/\gamma) + \varphi$. To satisfy the premise of Claim 7, it is required that $t = \gamma k/(2\mathcal{K}) \geq 8(c+1)\ln n$. To set the value of φ we consider the following cases.

Case 1 ($k \geq \sqrt{n/\gamma}\ln n$). Set $\varphi = \lceil 32(c+1)\sqrt{n/\gamma} \rceil$. Since $t = \frac{\gamma k}{2\lceil \frac{n}{\varphi} \rceil} \geq \frac{\gamma k}{4n/\varphi} \geq 8(c+1)\ln n$, then t satisfies the premise of Claim 7 and $\mathcal{Q} \leq k/t + \varphi = \frac{k}{\frac{\gamma k}{2\lceil \frac{n}{\varphi} \rceil}} + \varphi = \frac{2\lceil n/\varphi \rceil}{\gamma} + \varphi = O(\sqrt{n/\gamma})$.

Case 2 ($32(c+1)\ln n/\gamma < k < \sqrt{n/\gamma}\ln n$). Set $\varphi = \lceil 32(c+1)n\ln n/(\gamma k) \rceil$. Then $t = \frac{\gamma k}{2\lceil \frac{n}{\varphi} \rceil} \geq \frac{\gamma k}{4n/\varphi} \geq 8(c+1)\ln n$, satisfying Claim 7. Also $k/t \leq \sqrt{n/\gamma}\ln n/t = O(\sqrt{n/\gamma})$. Thus, $\mathcal{Q} \leq k/t + \varphi = O(\sqrt{n/\gamma} + \frac{n\ln n}{\gamma k})$.

Case 3 ($k \leq 32(c+1)\ln n/\gamma$). Each peer queries all n bits, resulting in $\mathcal{Q} = n$.

Since the protocol can check to see which case it is in (by inspecting k, n, and γ), the overall query complexity is $\mathcal{Q} = O(\min\{\sqrt{n/\gamma} + n\ln n/(\gamma k), n\})$.

C Proof of Lemma 9

The proof of Lemma 9 follows the next lemma.

▶ **Lemma 13.** *For every step $i \in [0, \lceil \lg(n/\varphi) \rceil]$, every interval in step i (those of size φ_i) is picked by at least t_i honest peers w.h.p*

Proof. Fix Step i. The number of intervals in step i is $\mathcal{K}_i$. The expected number of honest peers which pick a given interval at Step i is $E_i = \gamma k/\mathcal{K}_i = (\gamma k/(n/(2^i\varphi))) \geq 2^i(8(c+1)\ln n)$. Setting $t_i = E_i/2$, the probability of failure for any one interval, as given by Chernoff bounds (see previous subsection) is no more than $e^{-E_i/8} \leq n^{-c+1}$.

Taking a union bound over the $\sum_{i=0}^{\lg(n/\varphi)} \mathcal{K}_i < \sum_{i=0}^{\infty} \lceil \frac{n}{\varphi} \rceil \frac{1}{2^i} < n$ intervals over all steps i, the probability of any failure in any step is less than $n \cdot n^{-(c+1)} \leq n^{-c}$ for $c \geq 1$. ◀

Proof of Lemma 9. By induction on the steps. The base case, step 0, is trivial since every honest peer that picks an interval queries it completely.

For step $i > 0$, consider the interval $\ell = \ell_\mu^i$ picked by peer μ in step i. During step i, μ splits ℓ into two subintervals ℓ_L, ℓ_R of size φ_{i-1}. By Lemma 13, the intervals ℓ_L, ℓ_R were each picked by a least t_{i-1} honest peers w.h.p. during step $i-1$, and by the inductive hypothesis, those peers know (and broadcast) the correct strings s_L, s_R respectively. For $u \in \{L, R\}$, let $\mathsf{FS}_u = \mathsf{FS}(S(\ell_u), t_{i-1})$ denote the t_{i-1} frequent sets constructed in step i for ℓ_u. Then, $s_L \in \mathsf{FS}_L$ and $s_R \in \mathsf{FS}_R$. Hence, the decision trees built for ℓ_L and ℓ_R will contain leaves with labels s_L, s_R respectively, which will then be returned correctly by Procedure Determine, implying that μ learns the correct bit string for the interval ℓ, $s_\ell = s_L s_R$. ◀

D Trade-off among time, queries, and message size

Here we use information theory to prove a lower bound on the trade-offs among these parameters, as in [16].

▶ **Lemma 14.** *A lower bound on the trade-off among time t, number of non-faulty peers γk, number of queries (per peer) Q and maximum message size (per round) ϕ for a deterministic protocol is given by the following:* $(\gamma k t)[\lceil \lg \phi \rceil + \phi] + Q \geq n$.

Proof. We observe that there are 2^n possible outputs; hence n bits of information must be communicated to any peer. Assume that the adversarially controlled peers do not send any messages. Since an honest peer receives no more than ϕ bits (per round) from each of the other $\gamma k - 1$ honest peers over t rounds, the information received by the peer can be represented by a sequence of $(\gamma k - 1)t$ consecutive intervals, one for each message received, ordered by peer identity and time. Each interval contains the bits of the message, preceded by a sequence of $\lceil \lg \phi \rceil$ bits giving the exact number of bits in the message. Note that the $\lceil \lg \phi \rceil$ bits that represent the exact number of bits in the message are required to be able to know where every interval starts and ends. Hence, the total information received by a peer can be described by no more than $((\gamma k - 1)t)(\lceil \lg \phi \rceil + \phi) + Q$ bits, and this must be at least n. ◀

▶ **Corollary 15.** *If $((\gamma k - 1)t)(\lceil \lg \phi \rceil + \phi) + Q \leq n - 1$ then a deterministic protocol outputs incorrect results for at least half of the possible 2^n possible inputs.*

For randomized protocols: Assume the randomness is global. Any random protocol can be viewed as tossing the random coins first and then executing a deterministic protocol selected by the random coins.

▷ Claim 16. Let A be a randomized protocol. If all the information that the peer learns which depends on the input can be described in a sequence of only $n - 1$ bits, then there is an input X such that $Pr(A(X) = X) \leq 1/2$.

Proof of Claim. Suppose to the contrary that for every input X, $Pr(A(X) = X) > 1/2$. Let p_i be the probability that A chooses deterministic protocol A_i. Let $1_{X,i} = 1$ if $A_i(X) = X$. Then $\sum_{X \in \{0,1\}^n} \sum_i p_i 1_{X,i} > 2^n * (1/2) = 2^{n-1}$.

But from the Corollary above, $|\{X \mid A_i(X) = X\}| \leq 2^{n-1}$. Rearranging the terms,

$$\sum_{X \in \{0,1\}^n} \sum_i p_i 1_{X,i} = \sum_i p_i \sum_X 1_{X,i} \leq \sum_i p_i 2^{n-1} = 2^{n-1},$$

giving a contradiction. ◁

It is easy to see that if one assumes global randomness, then the description length is the same as what's needed for a deterministic protocol. Hence, the same lower bound on the trade-off applies. That is,

▶ **Theorem 17.** *Suppose there is a randomized protocol with time t, number of non-faulty peers γk, number of queries (per peer) Q, and maximum message size (per round) ϕ. If on every input, its output is correct with a probability greater than 1/2, then*

$$((\gamma kt) - 1)[\lceil \lg \phi \rceil + \phi] + Q \geq n - 1.$$

Note that this holds true for a protocol with public randomness and a fortiori, with private randomness. Furthermore, the same analysis holds in the Broadcast model. Hence the result of Theorem 12 is optimal up to $\log n$ factors in terms of the trade-off among queries, time, and message size.

Amnesiac Flooding: Easy to Break, Hard to Escape

Henry Austin
Durham University, UK

Maximilien Gadouleau
Durham University, UK

George B. Mertzios
Durham University, UK

Amitabh Trehan
Durham University, UK

Abstract

Broadcast is a central problem in distributed computing. Recently, Hussak and Trehan [PODC'19/STACS'20/DC'23] proposed a stateless broadcasting protocol (Amnesiac Flooding), which was surprisingly proven to terminate in asymptotically optimal time (linear in the diameter of the network). However, it remains unclear: (i) Are there other stateless terminating broadcast algorithms with the desirable properties of Amnesiac Flooding, (ii) How robust is Amnesiac Flooding with respect to *faults*?

In this paper we make progress on both of these fronts. Under a reasonable restriction (obliviousness to message content) additional to the fault-free synchronous model, we prove that Amnesiac Flooding is the *only* strictly stateless deterministic protocol that can achieve terminating broadcast. We achieve this by identifying four natural properties of a terminating broadcast protocol that Amnesiac Flooding uniquely satisfies. In contrast, we prove that even minor relaxations of *any* of these four criteria allow the construction of other terminating broadcast protocols.

On the other hand, we prove that Amnesiac Flooding can become non-terminating or non-broadcasting, even if we allow just one node to drop a single message on a single edge in a single round. As a tool for proving this, we focus on the set of all *configurations* of transmissions between nodes in the network, and obtain a *dichotomy* characterizing the configurations, starting from which, Amnesiac Flooding terminates. Additionally, we characterise the structure of sets of Byzantine agents capable of forcing non-termination or non-broadcast of the protocol on arbitrary networks.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Mathematics of computing → Graph algorithms; Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Amnesiac flooding, Terminating protocol, Algorithm state, Stateless protocol, Flooding algorithm, Network algorithms, Graph theory, Termination, Communication, Broadcast

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.10

Related Version *Full Version*: `https://arxiv.org/abs/2502.06001` [2]

Funding *George B. Mertzios*: Supported by the EPSRC grant EP/P020372/1.
Amitabh Trehan: Supported by the EPSRC grant EP/P021247/1.

Acknowledgements We would like to thank Danial Maqbool (Durham University) for his insightful work with a precursor to the uniqueness result.

1 Introduction

The dissemination of information to disparate participants is a fundamental problem in both the construction and theory of distributed systems. A common strategy for solving this problem is to "broadcast", i.e. to transmit a piece of information initially held by one agent to all other agents in the system [1, 18, 21, 24, 25]. In fact, broadcast is not merely


© Henry Austin, Maximilien Gadouleau, George B. Mertzios, and Amitabh Trehan;
licensed under Creative Commons License CC-BY 4.0



a fundamental communication primitive in many models, but also underlies solutions to other fundamental problems such as leader election and wake-up. Given this essential role in the operation of distributed computer systems and the potential volume of broadcasts, an important consideration is simplifying the algorithms and minimizing the overhead required for each broadcast.

Within a synchronous setting, *Amnesiac Flooding* as introduced by Hussak and Trehan in 2019 [13, 14] eliminates the need of the standard flooding algorithm to store historical messages. The algorithm terminates in asymptotically optimal $O(D)$ time (for D the diameter of the network) and is stateless as agents are not required to hold any information between communication rounds. The algorithm in the fault-free synchronous message passing model is defined as follows:

▶ **Definition 1** (Amnesiac flooding algorithm, adapted from [16]). *Let $G = (V, E)$ be an undirected graph, with vertices V and edges E (representing a network where the vertices represent the nodes of the network and edges represent the connections between the nodes). Computation proceeds in synchronous "rounds" where each round consists of nodes receiving messages sent from their neighbours. A receiving node then sends messages to some neighbours which arrive in the next round. No messages are lost in transit. The algorithm is defined by the following rules:*

- **(i)** *All nodes from a subset of sources or initial nodes $I \subseteq V$ send a message M to all of their neighbours in round 1.*
- **(ii)** *In subsequent rounds, every node that received M from a neighbour in the previous round, sends M to all, and only, those nodes from which it did not receive M. Flooding* terminates *when M is no longer sent to any node in the network.*

These rules imply several other desirable properties. Firstly, the algorithm only requires the ability to forward the messages, but does not read the content (or even the header information) of any message to make routing decisions. Secondly, the algorithm only makes use of local information and does not require knowledge of a unique identifier. Thirdly, once the broadcast has begun, the initial broadcaster may immediately forget that they started it.

However, extending Amnesiac Flooding and other stateless flooding algorithms (such as those proposed in [27, 29, 4]) beyond synchronous fault-free scenarios is challenging. This is due to the fragility of these algorithms and their inability to build in complex fault-tolerance due to the absence of state and longer term memory. It has subsequently been shown that no stateless flooding protocol can terminate under moderate asynchrony, unless it is allowed to perpetually modify a super-constant number (i.e. $\omega(1)$) of bits in each message [27]. Yet, given the fundamental role of broadcast in distributed computing, the resilience of these protocols is extremely important even on synchronous networks.

Outside of a partial robustness to crash failures, the fault sensitivity of Amnesiac Flooding under synchrony has not been explored in the literature. This omission is further compounded by the use of Amnesiac Flooding as an underlying subroutine for the construction of other broadcast protocols. In particular, multiple attempts have been made to extend Amnesiac Flooding to new settings (for example routing multiple concurrent broadcasts [4] or flooding networks without guaranteed edge availability [29]), while maintaining its desirable properties. However, none have been entirely successful, typically requiring some state-fulness. It has not in fact been established that *any other* protocol can retain all of Amnesiac Flooding's remarkable properties even in its original setting. These gaps stem fundamentally from the currently limited knowledge of the dynamics of Amnesiac Flooding beyond the fact of its termination and its speed to do so. In particular, all of the existing techniques (e.g. parity

arguments such as in [16] and auxiliary graph constructions such as in [28]) used to obtain termination results for Amnesiac Flooding are unable to consider faulty executions of the protocol and fail to capture the underlying structures driving terminating behaviour.

We address these gaps through the application of novel analysis and by considering the structural properties of Amnesiac Flooding directly. By considering the sequence of message configurations, we are able to identify the structures underlying Amnesiac Flooding's termination and use these to reason about the algorithm in arbitrary configurations. The resulting dichotomy gives a comprehensive and structured understanding of termination in Amnesiac Flooding. For example, we apply this to investigate the sensitivity of Amnesiac Flooding with respect to several forms of fault and find it to be quite fragile. Furthermore, we show that under reasonable assumptions on the properties of a synchronous network, any strictly stateless deterministic terminating broadcast algorithm oblivious to the content of messages, must produce the exact same sequence of message configurations as Amnesiac Flooding on any network from any initiator. We therefore argue that Amnesiac Flooding is unique. However, we show that if any of these restrictions are relaxed, even slightly, distinct terminating broadcast algorithms can be obtained. As a result of this uniqueness and simplicity, we argue that Amnesiac Flooding represents a prototypical broadcast algorithm. This leaves open the natural question: do there exist fundamental stateless algorithms underlying solutions to other canonical distributed network problems? Though memory can be essential or naturally useful in certain scenarios [5, 7, 8, 10, 17, 20], understanding what we can do with statelessness can help us push fundamental boundaries.

1.1 Our Contributions

In this work, we investigate the existence of other protocols possessing the following four desirable properties of Amnesiac Flooding:

1. *Strict Statelessness*: Nodes maintain no information other than their port labellings between rounds. This includes whether or not they were in the initiator set.
2. *Obliviousness*: Routing decisions may not depend on the contents of received messages.
3. *Determinism*: All decisions made by a node must be deterministic.
4. *Unit Bandwidth*: Each node may send at most one message per edge per round.

Our main technical results regarding the existence of alternative protocols to Amnesiac Flooding are given in the next two theorems (reworded in Section 4).

▶ **Theorem 2** (Uniqueness of Amnesiac Flooding). *Any terminating broadcast algorithm possessing all of* Strict Statelessness*,* Obliviousness*,* Determinism *and* Unit Bandwidth *behaves identically to Amnesiac Flooding on all graphs under all valid labellings for all source nodes.*

Note that the last theorem allows, but does not require, that nodes have access to unique identifiers labelling themselves and their ports. However, we enforce the condition that these identifiers, should they exist, may be drawn adversarially from some super set of $[n + \kappa]$ where n is the number of nodes on the networks and $\kappa = R(9, 8)$ where $R(9, 8)$ is a Ramsey Number (≤ 2662)[22]. It is important to stress here that Theorem 2 holds even if the space of unique identifiers is only greater than n by an additive constant. In contrast to the last theorem, the next one does not assume that agents have access to unique identifiers.

▶ **Theorem 3** (Existence of relaxed Algorithms). *There exist terminating broadcast algorithms which behave distinctly from Amnesiac Flooding on infinitely many networks possessing any three of:* Strict Statelessness*,* Obliviousness*,* Determinism *and* Unit Bandwidth*.*

We derive four of these relaxed algorithms which all build upon Amnesiac Flooding:

1. Neighbourhood-2 Flooding: *Strict Statelessness* is relaxed and agents are given knowledge of their neighbours' neighbours. This allows for distinct behaviour on networks of radius one as some agents are aware of the entire network topology.
2. Random Flooding: *Determinism* is relaxed and agents are given access to a random coin. Agents randomly choose in each round whether to use Amnesiac Flooding or to forward messages to all of their neighbours.
3. 1-Bit Flooding (Message Dependent): *Obliviousness* is relaxed and the source is allowed to include one bit of read-only control information in the messages. The source records in the message whether they are a leaf vertex, and if so agents perform Amnesiac Flooding upon receiving the message. Otherwise, agents perform a modified version where leaf vertices return messages.
4. 1-Bit Flooding (High Bandwidth): *Unit Bandwidth* is relaxed and agents are allowed to send either one or two messages over an edge. This permits the same algorithm as the previous case by encoding the control information in the number of messages sent.

We note that despite being a non-deterministic algorithm Random Flooding achieves broadcast with certainty and terminates almost surely in finite time.

We also perform a comprehensive investigation of the fault sensitivity of Amnesiac Flooding in a synchronous setting. Through the use of a method of invariants, we obtain stronger characterizations of termination than were previously known, for both Amnesiac Flooding, and a subsequently proposed Stateless Flooding protocol [27]. This allows us to provide precise characterizations of the behaviour of Amnesiac Flooding under the loss of single messages, uni-directional link failure, and time bounded Byzantine failures. The above invariants may be of independent interest, beyond fault sensitivity, as they provide strong intuition for how asynchrony interferes with the termination of both Amnesiac Flooding and the Stateless Flooding proposed in [27]. The main technical result concerning fault sensitivity is a *dichotomy* characterizing the configurations, starting from which, Amnesiac Flooding terminates. As the rigorous statement of the result requires some additional notation and terminology, we will state it only informally here. We show in Theorem 15 that, whether Amnesiac Flooding terminates when begun from a configuration or not, is determined exclusively by the parity of messages distributed around cycles and so-called faux-even cycles (FEC) (essentially pairs of disjoint odd cycles connected by a path, see section 5 for a full definition) within the graph. It follows from this characterisation that Amnesiac Flooding terminates from a configuration if and only if it the configuration can be reached from some sequence of multi-casts. Theorem 15 also implies the following three theorems which demonstrate the fragility of Amnesiac Flooding under three increasingly strong forms of fault. We give our fault model explicitly in Section 5.2.

▶ **Theorem 4** (Single Message Failure). *If single-source Amnesiac Flooding experiences a* single message drop failure *for the message* (u, v) *then it fails to terminate if and only if either or both of the following hold:*

- *uv is not a bridge*
- *uv lies on a path between vertex disjoint odd cycles*

Moreover, it fails to broadcast if and only if this is the first message sent along uv, uv is a bridge, and the side of the cut containing u does not contain an odd cycle.

We also consider the possibility of a link/edge failing in one direction (or if we consider an undirected edge as two directed links in opposite directions, only one of the directed links fails).

▶ **Theorem 5** (Uni-directional link failure). *For any graph $G = (V, E)$ and any initiator set $I \subsetneq V$ there exists an edge $e \in E$ such that a uni-directional link failure at e will cause Amnesiac Flooding to either fail to broadcast or fail to terminate when initiated from I on G. Furthermore, for any non-empty set of uni-directional link failures there exists $v \in V$ such that, when Amnesiac Flooding is initiated at v, it will either fail to broadcast or fail to terminate.*

We, finally, consider a set of Byzantine nodes, who know the original message, are free to collude among themselves and may decide to forward this message in any arbitrary pattern to their neighbours. However, for a discussion of termination to be meaningful, we require that the nodes have Byzantine behaviour for only a finite number of rounds.

▶ **Theorem 6** (Byzantine Failure). *If Amnesiac Flooding on $G = (V, E)$ initiated from $I \subsetneq V$ experiences a* weak Byzantine failure *at $J \subseteq V \setminus I$, then the adversary can force:*

- *Failure to broadcast if and only if J contains a cut vertex set.*
- *Non-termination if and only if at least one member of J lies on either a cycle or a path between odd-cycles.*

Two natural corollaries of the Theorem 4 we wish to highlight here are: (i) on any network from any initial node there exists a single message, the dropping of which, will produce either non-termination or non-broadcast and (ii) dropping any message on a bipartite network will cause either non-termination or non-broadcast. The latter requires the additional observation that each edge is traversed by a message precisely once in a bipartite graph under Amnesiac Flooding. Similarly, Theorem 6 implies that any Byzantine set containing a non-leaf node on any network can force either non-termination or non-broadcast for Amnesiac Flooding with any initial set.

1.2 Organisation of the paper

The initial part of the paper presents the required technical and motivational background, statements of our results and a technical outline of the more interesting proofs. Due to space limitations detailed proofs are deferred to the appendices and the full version [2]. Related work is presented in Section 2. Section 3 presents the model and notation required for the following technical sections. *Uniqueness* of the algorithm is discussed in Section 4. In subsection 4.1 we discuss the conditions under which the algorithm is unique and in subsection 4.2 we relax the conditions individually to derive additional algorithms. The proof of the uniqueness result is given in Appendix A. We present our work on the *termination dichotomy* for Amnesiac Flooding and its applications to *fault sensitivity* in Section 5 with the proof of the dichotomy given in Appendix B. We end with our conclusions and pointers to future work in Section 6.

2 Related Work

The literature surrounding both the broadcast problem and fault sensitivity is vast, and a summary of even their intersection is well beyond the scope of this work. Instead, we shall focus on work specifically concerning stateless, or nearly stateless broadcast.

The termination of Amnesiac Flooding and its derivatives has been the focus of several works. In combination they provide the following result that Amnesiac Flooding terminates under any sequence of multi-casts, i.e.

▶ **Theorem 7** (Termination of Amnesiac Flooding (adapted from [15, 16, 28])). *For $G = (V, E)$ a graph and $I_1, ..., I_k \subseteq V$ a sequence of sets of initiator nodes, Amnesiac Flooding on G terminates when initiated from I_1 in round 1, then I_2 in round 2 and so forth.*

Two independent proofs of the algorithm's termination have been presented, using either parity arguments over message return times [16] or axillary graph constructions [28]. The latter technique has further been used to establish tight diameter independent bounds on the termination of multicast using Amnesiac Flooding, complementing the eccentricity based bounds of [16]. The techniques we develop in this work, however, are more closely aligned with those of [15], as we exploit a similar notion to their "even flooding cycles" in our message path argument. In contrast to that work we focus on arbitrary configurations of messages, rather than just those resulting purely from a correct broadcast which allows us to obtain a stronger characterisation. Combining our techniques with the dual "reverse" flood introduced by [15], we are able to show the complement of Theorem 7, that only those configurations reachable from a sequence of multi-casts lead to termination.

There have, further, been multiple variants of Amnesiac Flooding introduced. It was observed by [27] in a result reminiscent of the BASIC protocol proposed by [12], that sending a second wave of messages from a subset of the initial nodes reduces the worst case $2D + 1$ termination time to the optimal $D + 1$ in all but a specific subset of bipartite graphs. We note that our fault sensitivity results extend naturally to this algorithm as well, as the same invariants apply to this setting. Beyond this, there have been several approaches to deal with the flooding of multiple messages simultaneously. In [15], the original authors of [13] show that under certain conditions termination can be retained, even when conflicting floods occur. Since then, two partially stateless algorithms have been proposed, both making use of message buffers and a small amount of local memory [29, 4]. We will not be directly concerned with these approaches, however, as we assume a single concurrent broadcast throughout. However, the mechanism employed in [29] should be highlighted as it rather cleverly exploits the underlying parity properties we identify as driving termination. Furthermore, as reduction to Amnesiac Flooding is used as a technique for proofs in many of these works, the comprehensive understanding of its termination we present here could prove a powerful tool for future work in these areas.

While the robustness of Amnesiac flooding and its variants have been previously studied, this has been focused on two forms of fault. The first is the disappearance and reappearance of nodes and links. The termination of Amnesiac Flooding is robust to disappearance and vulnerable to reappearance [16]. We will observe that this is a necessary consequence of the invariants driving termination and their relation to cyclicity. In particular, the disappearance of nodes and links cannot form new cycles violating the invariant, whereas their reappearance can. A pseudo-stateless extension to Amnesiac Flooding has been proposed to circumvent this [29], implicitly exploiting the parity conditions of [16]. The second are faults that violate synchrony. Under a strong form of asynchrony, truly stateless and terminating broadcast is impossible [27]. However, the landscape under a weaker form of asynchrony (namely, the case of fixed delays on communication links) is more fine-grained. Although termination results have been obtained for cycles, as well as the case of single delayed edges in bipartite graphs [16], there is no clear understanding of the impact of fixed channel delays. While we do not directly address this, we believe that techniques mirroring our invariant

characterizations may prove fruitful in this area. To our knowledge, this work is the first to consider both the uniqueness of Amnesiac Flooding, as well as its fault sensitivity beyond node/link unavailability in a synchronous setting.

Beyond Amnesiac Flooding and its extensions, the role of memory in information dissemination is well studied in a variety of contexts. Frequently, stateful methods obtain faster termination time, such as in the phone-call model where the ability to remember one's communication partners and prevent re-communication dramatically improves termination time and message efficiency [10, 5, 8]. Similarly, for bit dissemination in the passive communication model the addition of only $\log \log n$ bits of memory is sufficient to break the near linear time convergence lower bound of [7] and achieve polylogarithmic time [17]. Even more strongly, a recent work [20] has shown that in the context of synchronous anonymous dynamic networks, stabilizing broadcast from an idle start is impossible with $O(1)$ memory and even with $o(\log n)$ memory if termination detection is required. Despite this, low memory and even stateless broadcasts remain desirable [12]. The possibility of solving other canonical distributed computing problems beyond broadcast, in a stateless manner, remains intriguing. In this direction, various low memory stateful models have been proposed to handle more complex distributed problems e.g. the compact local streaming (CLS) model in [6] with deterministic solutions (routing, self-healing fault-tolerance etc.) and randomised solutions for distributed colouring in a similar model [11].

Stateless broadcast schemes have been studied in a variety of contexts. To give an example, they are used in mobile ad hoc networks, which, due to the lower power and rapid movement of devices, see diminishing returns from maintaining information about the network [19]. However, given a lack of synchronisation as well as the wish to avoid so-called broadcast storms [26], these techniques typically rely on either some form of global knowledge (such as the direction or distance to the initiator) or the ability to sample network properties by eavesdropping on communications over time [3, 23]. It should be noted however, that in contrast to many models, such as anonymous dynamic networks, radio networks and many mobile ad hoc networks, the typical framework for studying stateless flooding ("true statelessness" as defined by [27] restricting the model of [9]) permits the knowledge and distinguishing of neighbours in both broadcasting and receiving.

3 Model and Notation

Throughout this work we consider only finite, connected graphs on at least two nodes. We denote the set $\{1, ..., x\}$ by $[x]$ and $R(r, s)$ the Ramsey number such that any graph on $R(r, s)$ vertices contains either a clique on r vertices or an independent set on s vertices. In this work, we make use of a generic synchronous message passing model (as described in definition 1) with several additional assumptions based on the truly stateless model of [27]. Our agents are the nodes of a network and are able to communicate via messages of arbitrary size sent over the edges of this network. Computation occurs in synchronous rounds, consisting of three phases: (i) nodes receive messages sent in the previous round, (ii) nodes perform computation and (iii) nodes send messages to be delivered in the next round. Unless stated otherwise, agents do not suffer faults and no messages are lost. In addition to these standard assumptions, we enforce that the model is stateless, i.e. nodes cannot maintain any additional information between rounds (such as routing information, previous participation in the flood or even a clock value), cannot hold onto messages and can only forward, not modify the messages.

Unless stated otherwise we do not assume that nodes have access to unique identifiers, however they have locally labelled ports that are distinguishable and totally orderable for both receiving and sending messages. When we do work with identifiers these identifiers are assumed to be unique, drawn from $[|V| + \kappa]$ for $\kappa > 0$ a constant and assigned adversarially. We refer to such an assignment of IDs to a network as a labelling of the network and we identify the port label of a link leading to a node with the ID of that node. We further assume that individual nodes have access to arbitrarily powerful computation on information they do have.

We are principally interested in the problem of *broadcast*, although we will occasionally consider the related *multicast* problem i.e. there are multiple initiators who may potentially wake up in different rounds with the same message to be broadcast. For a graph $G = (V, E)$ and an *initiator* set $I \subseteq V$ we say that a node is informed if it has ever received a message from a previously informed node (where initiators are assumed to begin informed). An algorithm *correctly* solves broadcast (resp. multicast) on G if for all singleton (resp. non-empty) initiator sets there exists a finite number of rounds after which all nodes will be informed. Unless specified otherwise, we assume that initiator nodes remain aware of their membership for only a single round. We say that an algorithm *terminates* on $G = (V, E)$ if, for all valid initiator sets, there exists a finite round after which no further messages are sent (i.e. the communication network quiesces).
Formally, for the message M, we denote a *configuration* of Amnesiac Flooding as follows:

▶ **Definition 8.** *A configuration of Amnesiac Flooding on graph $G = (V, E)$ is a collection of messages/edges $S \subseteq \{(u, v) | uv \in E\}$ where $(u, v) \in S$ implies that in the current round u sent a message to v.*

Further, for H a subgraph of G, we denote by S_H, S restricted to H. Below, we define the operator $A_{I,G} : 2^{V^2} \to 2^{V^2}$ to implement one round of Amnesiac Flooding on the given configuration where agents in I receive the message from outside the network:

▶ **Definition 9.** *The operator $A_{I,G} : 2^{V^2} \to 2^{V^2}$ is defined as follows: The set I of nodes initiate the broadcast. The message $(u, v) \in A_{I,G}(S)$ if $uv \in E$, $(v, u) \notin S$ and either $\exists w \in V : (w, u) \in S$ or $u \in I$.*

We will drop the subscript when G is obvious from context and $I = \emptyset$. Further, we adopt the standard convention of using $A^k_{I,G}$ to mean $A_{I,G}$ applied k times. Theorem 7 in this notation can be expressed as follows:

▶ **Theorem 10** (Theorem 7 restated)**.** *For any graph $G = (V, E)$, and any finite sequence $I_1, ..., I_k \subseteq V$, there exists $m \in \mathbb{N}$ such that*

$$A^m_{\emptyset,G}(A_{I_k,G}(...A_{I_1,G}(\emptyset))) = \emptyset.$$

4 Uniqueness

In this section, we investigate broadcast protocols similar to Amnesiac Flooding and establish four desirable properties that Amnesiac Flooding uniquely satisfies in combination. On the other hand, we show that this result is sharp and that by relaxing any of these conditions one can obtain similar terminating broadcast protocols.

4.1 Uniqueness

Our first major result concerns the uniqueness of Amnesiac Flooding. Given the algorithms surprising properties, a natural question is whether other broadcast algorithms exist maintaining these properties. Specifically, does there exist a terminating protocol for broadcast which obeys all of the following for all graphs and valid port labellings:

1. *Strict Statelessness*: Nodes maintain no information other than their port labellings between rounds. This includes whether or not they were in the initiator set.
2. *Obliviousness*: Routing decisions may not depend on the contents of received messages.
3. *Determinism*: All decisions made by a node must be deterministic.
4. *Unit Bandwidth*: Each node may send at most one message per edge per round.

The answer is negative. We will actually prove the slightly stronger case, that this holds even if agents are provided with unique identifiers, are aware of the identifiers of their neighbours and that these identifiers have bounded size. Intuitively, the *Strict Statelessness* condition forces any broadcast protocol to make its forwarding decisions based only on the messages it receives in a given round. The combination of *Obliviousness* and *Unit Bandwidth* forces any protocol meeting the conditions to view messages as atomic. Finally, *Determinism* forces the protocol to make identical decisions every time it receives the same set of messages. Formally, any broadcast protocol meeting the four conditions must be expressible in the following form:

▶ **Definition 11.** *A* protocol $P = (b, f)$ *is a pair of functions, an* initial function b *and a* forwarding function f*, where* $b : \mathbb{N} \times 2^{\mathbb{N}} \to 2^{\mathbb{N}}$ *and* $f : \mathbb{N} \times 2^{\mathbb{N}} \times 2^{\mathbb{N}} \to 2^{\mathbb{N}}$*. The protocol is implemented as follows. On the first round the initiator node* s *with neighbourhood* $N(s)$ *sends messages to every node with a label in* $b(s, N(s))$*. On future rounds, each node* u *sends messages to every node with a label in* $f(u, N(u), S)$ *where* S *is the set of labels of nodes* u *received messages from in the previous round. Further, we require that for any* $B \subseteq A \subset \mathbb{N}$ $b(u, A), f(u, A, B) \subseteq A$ *and that* $f(u, A, \emptyset) = \emptyset$*, enforcing that agents can only communicate over edges of the graph and can only forward messages they have actually received respectively.*

In this setting, achieving broadcast is equivalent to every node receiving a message at least once and terminating in finite time corresponds to there existing a finite round after which no messages are sent. For example, Amnesiac Flooding is defined by the following functions:

▶ **Definition 12** (Amnesiac Flooding Redefinition)**.** *Amnesiac Flooding is defined by* $P_{AF} = (b_{AF}, f_{AF})$ *where, for all* $T \subseteq S \subset \mathbb{N}$*,* $b(u, S) = S$ *and* $f(u, S, T) = S \setminus T$ *if* $T \neq \emptyset$ *and* $\emptyset$ *otherwise.*

In order, to argue that Amnesiac Flooding is unique we require a notion of what it means for two broadcast algorithms to be distinct.

▶ **Definition 13.** *Let* $G = (V, E)$ *be a graph and* $L \subseteq \mathbb{N}$ *a set of labels. We say that the pair* (G, L) distinguishes *the protocols* P *and* Q *if there exists a labelling of* G *using only labels from* L *such that for some initial vertex* s*,* P *and* Q *send messages over different sets of edges in the same round when implementing broadcast on* G *initiated from* s*. If there exists some pair* (G, L) *which distinguishes* P *and* Q*, we describe* P *and* Q *as* distinct*. Otherwise, we consider them the same protocol.*

For a protocol $P = (b, f)$, a set $S \subseteq \mathbb{N}$ and a number $k \in \mathbb{N}$ we describe P as AF up to degree k on S if there is no graph G of maximum degree k such that (G, S) distinguishes P from P_{AF}. From here on we will assume that all unique labels are drawn from $[n + \kappa]$

where κ is a sufficiently large constant and n is the number of nodes in the graph. As this only eliminates possible pairs distinguishing protocols from Amnesiac Flooding, this only strengthens the result. We obtain the following result, discussion and sketching the proof of which makes up the remainder of this section, provided $\kappa \geq R(9, 8)$.

▶ **Theorem 14.** *(Restatement of Theorem 2) Let $P = (b, f)$ be a correct and terminating broadcast protocol defined according to definition 11, then P is not distinct from Amnesiac Flooding.*

Proof sketch for Theorem 14. The basic argument is to show that any correct and terminating broadcast protocol meeting the criteria is identical to Amnesiac Flooding. Our core technique is to construct a set of network topologies such that any policy distinct from Amnesiac Flooding fails on at least one of them. However, the behaviour of these alternative protocols can be quite complex, as can the relationship between the constraints enforced by different networks under different labellings. In order to circumvent this, we find very simple networks, and labellings there of, where any algorithm must behave like Amnesiac Flooding and then modify them to obtain new instances, with the property that only a small number of vertices may ever behave distinctly from Amnesiac Flooding. In these more manageable cases, we are able to then show that any distinct behaviour leads to an incorrect algorithm. More precisely, for any given protocol we derive a directed graph (separate from the network topology) describing its behaviour and demonstrate via a forbidden subgraph argument that any set of IDs of size $R(x, 8)$ must contain a subset T of size at least x such that P is AF up to degree 1 on T. We take $\kappa = R(9, 8)$ and show that there must then exist $U \subseteq T$ containing at least 6 identifiers such that P is AF up to degree 2 on U. By constructing a set of small sub-cubic graphs, we are able to extend this to degree 3.
These form the base case of a pair of inductive arguments. First, we construct a progression of sub-cubic graphs which enforce that if P is AF on $[m]$ up to degree 3 it must be AF on $[m + 1]$ up to degree 3. We then construct a family of graphs which have a single node of high-degree, while all other nodes have a maximum degree of 3 and so must behave as though running Amnesiac Flooding. These graphs permit a second inductive argument showing that this unique high degree node must also behave as if running Amnesiac Flooding. In combination, these two constructions enforce that P behaves like Amnesiac Flooding in all possible cases. The proof is given in appendix A. ◀

4.2 Relaxing the constraints

Despite the uniqueness established in the previous subsection, we are able to derive four relaxed algorithms distinct from Amnesiac Flooding each obeying only three of the four conditions.

▶ **Theorem 3** (Existence of relaxed Algorithms)**.** *There exist terminating broadcast algorithms which behave distinctly from Amnesiac Flooding on infinitely many networks possessing any three of:* Strict Statelessness*,* Obliviousness*,* Determinism *and* Unit Bandwidth*.*

The algorithms we obtain all build upon Amnesiac Flooding, and we believe illuminate the role of each of the four conditions in the uniqueness result by showing what they prevent. The algorithms for *Obliviousness* and *Unit Bandwidth* are presented together, as they differ only in how control information is encoded.

- *Strict Statelessness*: Several relaxations of this already exist, such as Stateless Flooding (the initiator retains information for one round) or even classical non-Amnesiac Flooding (nodes are able to retain 1-bit for one round). We present Neighbourhood-2 Flooding. Nodes know the ID of their neighbours' neighbours. The protocol behaves distinctly on star graphs, as the hub can determine the entire network topology.

- *Obliviousness* and *Unit Bandwidth*: 1-BIT FLOODING. Nodes are allowed to send a single bit of read-only control information (in the message header or encoded in the number of messages sent) communicating whether the initiator is a leaf vertex. If it is, nodes implement Amnesiac Flooding, otherwise they use a different mechanism called PARROT FLOODING (leaves bounce the message back) which always terminates when begun from a non-leaf vertex.
- *Determinism*: RANDOM-FLOODING. Nodes have access to one bit of randomness per round. Each round every node randomly chooses to implement Amnesiac Flooding or to forward to all neighbours. RANDOM-FLOODING is correct with certainty and terminates almost surely in finite time.

Since each of these constitutes only a minor relaxation of the restrictions, we argue that the uniqueness of Amnesiac Flooding is in some sense sharp. We present the protocols fully and demonstrate their correctness and termination for each of these cases independently in the full version [2].

5 Termination Dichotomy

With the uniqueness of Amnesiac Flooding established, a greater understanding of its properties is warranted. In this section, we study the configuration space of Amnesiac Flooding and obtain an exact characterisation of terminating configurations. We then apply this to investigate the algorithm's fault sensitivity.

5.1 Obtaining a termination dichotomy

In order to consider the fault sensitivity of Amnesiac Flooding, we need to be able to determine its behaviour outside of correct broadcasts. Unfortunately, neither of the existing termination proofs naturally extend to the case of arbitrary message configurations. Fortunately, we can derive an invariant property of message configurations when restricted to subgraphs that exactly captures non-termination, which we will call "balance" (see Definition 20).

▶ **Theorem 15.** *Let S be a configuration on $G = (V, E)$ then there exists $k \geq 0$ such that $A^k_G(S) = \emptyset$ if and only if S is balanced on G.*

In fact we obtain that not only do balanced configurations terminate, they terminate quickly.

▶ **Corollary 16.** *Let S be a balanced configuration on $G = (V, E)$ then there exists $k \leq 2|E|$ such that $A^k_G(S) = \emptyset$.*

Intuitively, for the protocol not to terminate, we require that a message is passed around forever and since it is impossible for a message to be passed back from a leaf node the message must traverse either a cycle or system of interconnected cycles. As we will demonstrate in the rest of the section we need only consider systems of at most two cycles. Specifically, we introduce an invariant property determined by parity constraints on the number of messages travelling in each direction and their spacing around: odd-cycles, even cycles and what we will dub, faux-even cycles.

▶ **Definition 17.** *A* faux-even cycle *(FEC) is a graph comprised of either two node disjoint odd cycles connected by a path or two odd cycles sharing only a single node. We denote by $FEC_{x,y,z}$ the FEC with one cycle of length $2x+1$, one of length $2z+1$ and a path containing y edges between them. We emphasize that if $y = 0$ the two cycles share a common node and if $y = 1$ the two cycles are connected by a single edge.*

FECs get their name from behaving like even cycles with respect to the operator A. In order to capture this we can perform a transformation to convert them into an equivalent even cycle.

▶ **Definition 18.** *Let $F = (V, E)$ be $FEC_{x,y,z}$. Then the* even cycle representation *of F denoted F_2 is the graph constructed by splitting the end points of the interconnecting path in two, and duplicating the path to produce an even cycle. Formally, if the two cycles are of the form $a_0...a_{2x}a_0$ and $c_0...c_{2z}c_0$, with a_0 and c_0 connected by the path $b_1...b_{y-1}$, we construct the following large even cycle from four paths: $a_0...a_{2x}a_{-1}d_1...d_{y-1}c_{-1}c_{2z}...c_0b_{y-1}...b_1a_0$. Here a_{-1} is a copy of a_0, c_{-1} is a copy of c_0, and the path $d_1...d_{y-1}$ is a copy of the path $b_1...b_{y-1}$ (See figure 1). Note that if $y = 0$, $a_0 = c_0$ and so we do not include any nodes from b or d. Similarly, if $y = 1$, a_0 and c_0 are connected by a single edge, as are a_{-1} and c_{-1}.*

There then exists a corresponding message configuration over the even cycle representation. Essentially (other than a few technical exceptions), this new configuration is the same as the old configuration but with two copies of each message on the path, one on each corresponding edge of the even cycle representation. Formally,

▶ **Definition 19.** *Let $F = (V, E)$ be $FEC_{x,y,z}$ and S a configuration of Amnesiac Flooding on F, the* even cycle representation *of S on F denoted $S_{2,F}$ is determined as follows. For each $m \in S$*

- *If $m = (a_{2x}, a_0)$ (resp. (a_0, a_{2x})) we add (a_{2x}, a_{-1}) (resp. (a_{-1}, a_{2x})) to $S_{2,F}$.*
- *If $m = (b_i, b_j)$ for some $i, j \in \{1, ..., y-1\}$, we add both (b_i, b_j) and (d_i, d_j) to $S_{2,F}$.*
- *If $m = (c_{2z}, c_0)$ (resp. (c_0, c_{2z})) we add (c_{2z}, c_{-1}) (resp. (c_{-1}, c_{2z})) to $S_{2,F}$.*
- *If $m = (a_0, b_1)$ (resp. (b_1, a_0)) we add both (a_0, b_1) and (a_{-1}, d_1) (resp. (b_1, a_0) and (d_1, a_{-1})) to $S_{2,F}$.*
- *If $m = (c_0, b_{y-1})$ (resp. (b_{y-1}, c_0)) we add both (c_0, b_{y-1}) and (c_{-1}, d_{y-1}) (resp. (b_{y-1}, c_0) and (d_{y-1}, c_{-1})) to $S_{2,F}$.*
- *If $m = (a_0, c_0)$ (resp. (c_0, a_0)) we add both (a_0, c_0) and (a_{-1}, c_{-1}) (resp. (c_0, a_0) and (c_{-1}, a_{-1})) to $S_{2,F}$.*
- *Otherwise we add m to $S_{2,F}$.*

With this established we can now define the notion of balance.

▶ **Definition 20.** *A configuration S is* balanced *on $G = (V, E)$ if for all subgraphs H of G one of the following holds:*

- *H is not a cycle or FEC.*
- *H is an odd cycle and S_H contains an equal number of messages travelling clockwise and anti-clockwise on H.*
- *H is an even cycle and for any given message m in S_H, there is an equal number of messages travelling clockwise and anti-clockwise on H such that their heads are an even distance from m's.*
- *H is an FEC and $S_{2,H}$ is balanced on H_2 (i.e. obeys the previous condition).*

With these definitions established, we can present the intuition behind the proof of Theorem 15.

Sketch of the proof of Theorem 15. We first establish that balance, and therefore imbalance, is conserved by Amnesiac flooding and, as the empty configuration is balanced, Amnesiac Flooding cannot terminate from any imbalanced configuration. For Amnesiac Flooding not to terminate it requires that some message travels around the communication graph and returns to the same edge, in the same direction. We show that if a configuration is balanced, the trajectory of any message can spend only a bounded number of consecutive steps on any

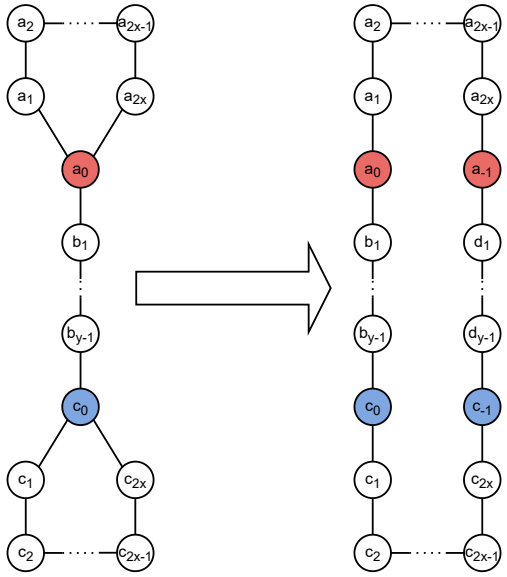


Figure 1 Left: An $FEC_{x,y,z}$. Right: The corresponding even cycle representation. Please note that this depiction only holds for $y \geq 2$. For $y = 1$: a_0 and c_0 are connected directly by an edge in both sub figures (as are a_{-1} and c_{-1}). For $y = 0$: $a_0 = c_0$ and $a_{-1} = c_{-1}$.

given cycle or FEC. However, we can also show that any message's trajectory which crosses the same edge twice in the same direction, must have spent a large number of consecutive steps on some cycle or FEC, and therefore could not have begun in a balanced configuration. Thus, Amnesiac Flooding started from any balanced configuration must terminate. The proof is given in appendix B. ◀

5.2 Applying the termination dichotomy

In this section, we apply the dichotomy to obtain a number of results.

5.2.1 Extended Dichotomy

While Theorem 15 provides a full dichotomy over the configuration space of Amnesiac Flooding and is easier to reason about than previous results, the definition is somewhat unwieldy. In this subsection, we demonstrate the effectiveness of the dichotomy and unify it with the existing results [15, 16, 28]. It has previously been observed that running Amnesiac Flooding backwards obtains another instance of multi-cast Amnesiac Flooding [15]. Formally,

▶ **Definition 21.** *Let $G = (V, E)$ be a graph and S be a configuration of messages on G. Then $\bar{S} = \{(v, u) | (u, v) \in S\}$.*

The following lemma is similar to the argument made in Corollary 4.6. of [15].

▶ **Lemma 22.** *Let $G = (V, E)$ be a graph, S a configuration of messages on G and $T = \{u \in V | \forall v \in N(u) : (u, v) \in S\}$ the set of source vertices. Then $A_{T,G}\left(\overline{A_{\emptyset,G}(\bar{S})}\right) = S$*

Which gives the following immediately via induction,

▶ **Lemma 23.** *Let $G = (V, E)$ be a graph and S a configuration of messages on G. Then for any $k \in \mathbb{N}$ there exists a sequence $I_1, ..., I_k \subseteq V$ such that $A_{I_1,G}\left(...\left(A_{I_k,G}\left(\overline{A^k_{\emptyset,G}\left(\bar{S}\right)}\right)\right)...\right) = S$.*

Intuitively, this means that given any configuration S of Amnesiac Flooding, we can run it backwards through time to some earlier configuration S'. Further we obtain a sequence of vertex sets $I_1, ..., I_k$ that were sinks in the time-reversed process and therefore sources in the forwards process. We can therefore reconstruct S beginning from S' via some sequence of fresh multi-casts from $I_1, ..., I_k$. We will use this fact to obtain all configurations from which Amnesiac Flooding terminates (i.e. balanced configurations) from the empty configuration. The following lemma is immediate from the definition of balance (definition 20), as reversing the direction of all messages in a configuration does not affect its balance.

▶ **Lemma 24.** *$\bar{S}$ is balanced on G if and only if S is balanced on G*

Putting it all together, we obtain the following extension of the dichotomy result, as well as the complement to Theorem 7.

▶ **Theorem 25.** *Let $G = (V, E)$ be a graph and S a configuration of G, the following are all equivalent:*

1. $\exists k \in \mathbb{N} : A^k_{\emptyset,G}(S) = \emptyset$
2. $\exists k \in \mathbb{N}, I_1, ..., I_k \subseteq V : A_{I_k,G}\left(... \left(A_{I_1,G}(\emptyset)\right) ...\right) = S$
3. *S is balanced on G*

Proof. The equivalence of (1) and (3) follow immediately from Theorem 15. Further, we have that (2) implies (1) from Theorem 7. Now assume S is balanced, then by lemma 24, so is $\bar{S}$. Thus by Theorem 15 there exists a finite k such that after k rounds Amnesiac flooding started from $\bar{S}$ must terminate, i.e. $A^k_{\emptyset,G}(\bar{S}) = \emptyset$. Therefore, by lemma 23 we have a sequence $I_1, ..., I_k \subseteq V$ such that $S = A_{I_1,G}\left(...\left(A_{I_k,G}\left(\overline{A^k_{\emptyset,G}\left(\bar{S}\right)}\right)\right)...\right) = A_{I_1,G}\left(... \left(A_{I_k,G}(\emptyset)\right) ...\right)$. Thus, we have (3) implies (2) and the result follows. ◀

5.2.2 Fault Sensitivity

In this work we consider three key forms of fault of increasing severity: message dropping, uni-directional link failure and weak-Byzantine failures. Intuitively, these correspond to a set of messages failing to send in a specific round, a link failing in one direction creating a directed edge and a set of nodes becoming transiently controlled by an adversary.

More precisely, let $\mathbf{S} = (S_i)_{i\in\mathbb{N}}$ be the sequence of actual message configurations on our network. We say that $\mathbf{S}$ is *fault free* for $G = (V, E)$ **if** $S_{i+1} = A_G(S_i)$ for all $i \in \mathbb{N}$. Otherwise, we say it experienced a *fault*. In this case we say $\mathbf{S}$ has suffered from,

- Message dropping, if there exists $T \subseteq V^2$ and $k \geq 1$ such that $S_{k+1} = A(S_k) \setminus T$ and for all $i \neq k$, $S_{i+1} = A(S_i)$. This corresponds to all messages in T being dropped on round k.
- Uni-directional link failure, if there exists $X \subseteq V^2$ such that for all $i \geq 1$, $S_{i+1} = A(S_i) \setminus X$. This corresponds to all oriented links in X failing.
- Weak-Byzantine failure, if there exists $Y \subseteq V$ such that for some k at least twice the diameter, for all $i < k$, $S_{i+1} \setminus \{(u, v)|u \in Y\} = A(S_i) \setminus \{(u, v)|u \in Y\}$. This corresponds to a possible failure where an adversary determines the forwarding decisions of the nodes in Y until round k.

Note that we refer to the Byzantine failures as weak, since they are transient and only interfere with the forwarding of the message, not its content. It is obvious to see that in a stateless setting there is no way to deal with a Byzantine fault that changes the message

as there is no method to verify which message is authentic. Intuitively, in our setting, Weak-Byzantine agents may choose to send messages to an arbitrary set of neighbours in each round and they are all controlled by a single coordinated adversary. We say that a Weak-Byzantine adversary with control of a given set of nodes can force some behaviour if there exists any weak byzantine failure on that set of nodes producing the forced behaviour. We can now express our fault sensitivity results, the proofs of which we defer to the full version [2], and begin with an extreme case of single message dropping.

▶ **Theorem 4** (Single Message Failure). *If single-source Amnesiac Flooding experiences a* single message drop failure *for the message* (u, v) *then it fails to terminate if and only if either or both of the following hold:*

- *uv is not a bridge*
- *uv lies on a path between vertex disjoint odd cycles*

Moreover, it fails to broadcast if and only if this is the first message sent along uv, uv is a bridge, and the side of the cut containing u does not contain an odd cycle.

Thus, Amnesiac Flooding is extremely fault-sensitive with respect to message dropping. Secondly, considering uni-directional link failures we obtain the following.

▶ **Theorem 5** (Uni-directional link failure). *For any graph $G = (V, E)$ and any initiator set $I \subsetneq V$ there exists an edge $e \in E$ such that a uni-directional link failure at e will cause Amnesiac Flooding to either fail to broadcast or fail to terminate when initiated from I on G. Furthermore, for any non-empty set of uni-directional link failures there exists $v \in V$ such that, when Amnesiac Flooding is initiated at v, it will either fail to broadcast or fail to terminate.*

Finally for the weak-Byzantine case.

▶ **Theorem 6** (Byzantine Failure). *If Amnesiac Flooding on $G = (V, E)$ initiated from $I \subsetneq V$ experiences a* weak Byzantine failure *at $J \subseteq V \setminus I$, then the adversary can force:*

- *Failure to broadcast if and only if J contains a cut vertex set.*
- *Non-termination if and only if at least one member of J lies on either a cycle or a path between odd-cycles.*

6 Conclusions and Future Work

In this paper, we prove a *uniqueness* result: Under standard synchronous message passing assumptions, any strictly stateless deterministic algorithm oblivious to the message content which solves terminating broadcast is indistinguishable from *Amnesiac Flooding*. We therefore argue due to both its uniqueness and simplicity, that Amnesiac Flooding is a fundamental or prototypical broadcast algorithm. We formalise the four properties required for this uniqueness to hold, and show that by relaxing each individually one can obtain other correct and terminating broadcast algorithms, of which we present several. These present the following natural questions: To what extent does Amnesiac Flooding represent a "minimal" broadcast algorithm? Are there identifiable families of algorithms solving terminating broadcast with a subset of these restrictions? Are any of these independent of (i.e. not derivatives of) Amnesiac Flooding? Lastly, we are not aware of any similar uniqueness results in algorithms literature, restricting the number of successful algorithms to just one (or even to small finite numbers) - can this result stimulate a study into enumerating distinct algorithms for solutions to interesting problems? Note that our model of true statelessness rules out trivial extensions to algorithms such as delaying algorithm start or holding messages for a certain count. Are there reasonable ways to define distinctiveness i.e. discard "trivial" extensions to algorithms in less restrictive models than ours?

We also obtain an understanding of the structural properties of Amnesiac Flooding. In particular, we study its sensitivity to single message drops, uni-directional link failures, and weak byzantine collusion, showing it can easily become non-terminating or non-broadcasting under such conditions. This is perhaps surprising, as statelessness is frequently associated with fault tolerance, such as in the self stabilizing setting. A reasonable interpretation of Theorem 15, however, is that Amnesiac Flooding, while locally stateless, depends heavily on a distributed "meta-state" contained in the configuration of sent messages. This suggests it is unlikely that any minor modification of Amnesiac Flooding will resolve its fragility without depending on an entirely different mechanism for termination. In support of this, we note that of the four alternatives presented in the proof of Theorem 3, only Random-Flooding is meaningfully more robust (and will in fact terminate from any configuration in finite time almost surely). Nevertheless, we contend that further exploration of stateless algorithms such as Amnesiac Flooding, their properties and related models are important for both theory and practice of distributed networks.

References

1 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.

2 Henry Austin, Maximilien Gadouleau, George B. Mertzios, and Amitabh Trehan. Amnesiac flooding: Easy to break, hard to escape, 2025. doi:10.48550/arXiv.2502.06001.

3 Abhik Banerjee, Chuan Heng Foh, Chai Kiat Yeo, and Bu-Sung Lee. Performance improvements for network-wide broadcast with instantaneous network information. *J. Netw. Comput. Appl.*, 35(3):1162–1174, 2012. doi:10.1016/J.JNCA.2012.01.008.

4 Zahra Bayramzadeh, Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Weak amnesiac flooding of multiple messages. In Karima Echihabi and Roland Meyer, editors, *Networked Systems - 9th International Conference, NETYS 2021, Virtual Event, May 19-21, 2021, Proceedings*, volume 12754 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2021. doi:10.1007/978-3-030-91014-3_6.

5 Petra Berenbrink, Robert Elsässer, and Thomas Sauerwald. Randomised broadcasting: Memory vs. randomness. *Theor. Comput. Sci.*, 520:27–42, 2014. doi:10.1016/J.TCS.2013.08.011.

6 Armando Castañeda, Jonas Lefèvre, and Amitabh Trehan. Fully compact routing in low memory self-healing trees. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *ICDCN 2020: 21st International Conference on Distributed Computing and Networking, Kolkata, India, January 4-7, 2020*, pages 21:1–21:10. ACM, 2020. doi:10.1145/3369740.3369786.

7 Niccolò D'Archivio and Robin Vacus. On the Limits of Information Spread by Memory-Less Agents. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.18.

8 Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 21–30, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993640.

9 Danny Dolev, Michael Erdmann, Neil Lutz, Michael Schapira, and Adva Zair. Stateless computation. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 419–421. ACM, 2017. doi:10.1145/3087801.3087854.

10 Robert Elsässer and Thomas Sauerwald. The power of memory in randomized broadcasting. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium*

on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008, pages 218–227. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347107.

11 Maxime Flin, Mohsen Ghaffari, Magnús M. Halldórsson, Fabian Kuhn, and Alexandre Nolin. Coloring fast with broadcasts. In Kunal Agrawal and Julian Shun, editors, *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023, Orlando, FL, USA, June 17-19, 2023*, pages 455–465. ACM, 2023. doi:10.1145/3558481.3591095.

12 Ajei S. Gopal, Inder S. Gopal, and Shay Kutten. Fast broadcast in high-speed networks. *IEEE/ACM Trans. Netw.*, 7(2):262–275, 1999. doi:10.1109/90.769773.

13 Walter Hussak and Amitabh Trehan. On termination of a flooding process. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 153–155. ACM, 2019. doi:10.1145/3293611.3331586.

14 Walter Hussak and Amitabh Trehan. On the termination of flooding. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 17:1–17:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.STACS.2020.17.

15 Walter Hussak and Amitabh Trehan. Terminating cases of flooding. *CoRR*, abs/2009.05776, 2020. arXiv:2009.05776.

16 Walter Hussak and Amitabh Trehan. Termination of amnesiac flooding. *Distributed Comput.*, 36(2):193–207, 2023. doi:10.1007/S00446-023-00448-Y.

17 Amos Korman and Robin Vacus. Early adapting to trends: self-stabilizing information spread using passive communication. *Distributed Comput.*, 37(4):335–362, 2024. doi:10.1007/S00446-024-00462-8.

18 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

19 Victoria Manfredi, Mark Crovella, and Jim Kurose. Understanding stateful vs stateless communication strategies for ad hoc networks. In Parmesh Ramanathan, Thyaga Nandagopal, and Brian Neil Levine, editors, *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking, MOBICOM 2011, Las Vegas, Nevada, USA, September 19-23, 2011*, pages 313–324. ACM, 2011. doi:10.1145/2030613.2030649.

20 Garrett Parzych and Joshua J. Daymude. Memory Lower Bounds and Impossibility Results for Anonymous Dynamic Broadcast. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.35.

21 David Peleg. *Distributed Computing: A Locality Sensitive Approach*. SIAM, 2000.

22 Stanisław Radziszowski. Small ramsey numbers. *The electronic journal of combinatorics*, pages DS1–Sep, 2012.

23 Patricia Ruiz and Pascal Bouvry. Survey on broadcast algorithms for mobile ad hoc networks. *ACM Comput. Surv.*, 48(1):8:1–8:35, 2015. doi:10.1145/2786005.

24 Andrew Tanenbaum. *Computer networks*. Pearson Prentice Hall, Boston, 2011.

25 Gerard Tel. *Introduction to distributed algorithms*. Cambridge University Press, New York, NY, USA, 1994.

26 Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wirel. Networks*, 8(2-3):153–167, 2002. doi:10.1023/A:1013763825347.

27 Volker Turau. Stateless information dissemination algorithms. In Andréa Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2020. doi:10.1007/978-3-030-54921-3_11.

28 Volker Turau. Amnesiac flooding: Synchronous stateless information dissemination. In Tomás Bures, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdzinski, Claus Pahl, Florian Sikora, and Prudence W. H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2021. doi:10.1007/978-3-030-67731-2_5.

29 Volker Turau. Synchronous concurrent broadcasts for intermittent channels with bounded capacities. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 296–312. Springer, 2021. doi:10.1007/978-3-030-79527-6_17.

Appendix

In this section we present the proof of our main technical results, i.e. theorems 14 and 15, as well as their constituent lemmas. Due to space constraints, we defer the proofs of all such lemmas as well as the correctness of all other results to the full version. Additionally, we will refer to Amnesiac Flooding as simply AF throughout.

A Proof of Uniqueness (Theorem 14)

We begin with the following observations, which follow from case analysis on paths:

▶ **Observation 26.**

1. *For all $\emptyset \neq S \subset \mathbb{N}$ and $u \in \mathbb{N}$, $b(u, S) \neq \emptyset$. Otherwise no new nodes will be informed after the first round, violating correctness.*
2. *For $u, v, w \in \mathbb{N}$, $f(v, \{u, w\}, \{u\}) \in \{\{w\}, \{u, w\}\}$. Otherwise, if uvw is a subsequence of a path the message started from u will not reach w.*
3. *For $u, v \in [\kappa]$, if $f(u, \{v\}, \{v\}) = v$ then $f(v, \{u\}, \{u\}) = \emptyset$. Otherwise P will not terminate on the two node path labelled with u and v.*
4. *For $u, v, w \in [\kappa]$ if $f(u, \{v\}, \{v\}) = \{v\}$ and $f(w, \{v\}, \{v\}) = \{v\}$ then $f(v, \{u, w\}, \{u\}) = \{u, w\}$ and/or $f(v, \{u, w\}, \{w\}) = \{u, w\}$. Additionally, $f(v, \{u, w\}, \{u, w\}) = \emptyset$. Otherwise the protocol would not terminate on the three node path with labels uvw.*

Via a combinatorial argument followed by case analysis, we show the existence of a small set of labels which when restricted to, P behaves identically to AF for subcubic graphs.

▶ **Lemma 27.** *There exists $T \subseteq [\kappa]$ such that $|T| \geq 6$ and P is AF on T up to degree 3.*

With this established we can extend the result to all labels via an inductive argument. Essentially, we are able to construct a sequence of graph-labelling pairs that grow to include all labels with all possible neighbourhoods. At each state, by our constructions and the induction hypothesis, at most four vertices may behave distinctly from AF. By careful construction, however, any distinct behaviour will lead to an incorrect algorithm.

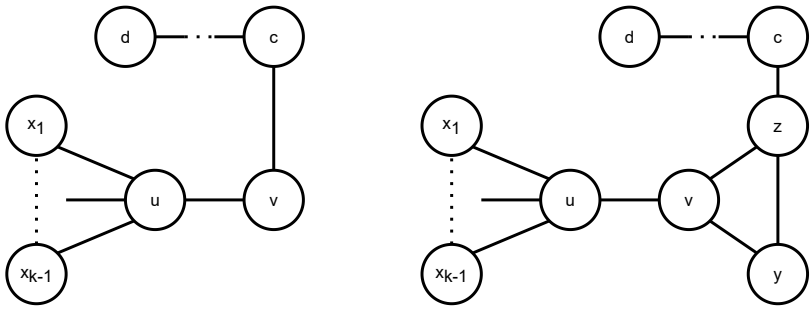
▶ **Lemma 28.** *P is AF on $\mathbb{N}$ up to degree 3.*

The main proof makes use of a very similar argument. However, we now construct graphs with one high degree vertex, while the rest all have degree at most three and so behave indistinguishably from AF.

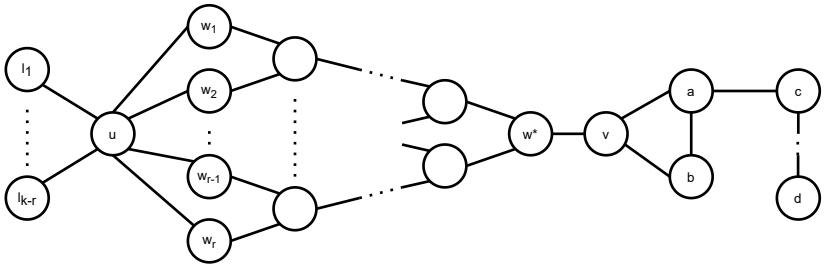
Proof of Theorem 14. By lemma 28 we have that P is AF up to degree 3 on $\mathbb{N}$. It is immediate that for all $u \in \mathbb{N}$, $S \subset \mathbb{N}$ $b(u, S) = S$ as u could be at the centre of a star with one of its leaves replaced by a long path. In this case, u will never receive a message again and so must send to all of its neighbours in the first round.

For an arbitrary label $u \in \mathbb{N}$ and degree $k \in \mathbb{N}$ we can show that u must behave as though it is implementing AF if it is at a node of degree k. Take $S \subset \mathbb{N}$ such that $|S| = k$, we will show that for all non-empty $T \subseteq S$, $f(u, S, T) = S \setminus T$ via induction on the size of T.

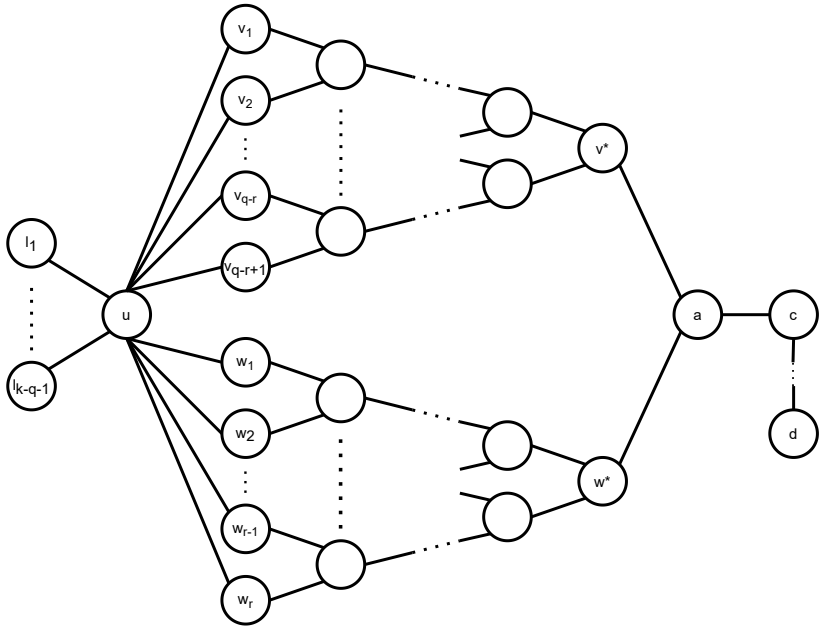
Beginning with our base case. If u receives from a single neighbour we can construct a pair of graphs (special cases of our general construction) that enforce the AF policy. Consider the tree in figure 2a, if u receives a message from v it must send to at least $x_1, ..., x_{k-1}$ as u will receive messages in only a single round. This follows as the rest of tree will use AF policies and since the graph is bipartite each node will be active only once. Thus, the only two options for $f(u, S, \{v\})$ are S or $S \setminus \{v\}$. Now consider the second graph from figure 2a and a broadcast initiated at u. We can see that if $f(u, S, \{v\}) = S$ then the cycle and u will simply pass a message back and forth forever. Thus, $f(u, S, \{v\}) = S \setminus \{v\}$.
We now generalize this construction and perform our induction. Assume that for any non-empty subset T of S of size at most q, $f(u, S, T) = S \setminus T$. For the sake of contradiction assume that this is not true for V where $T \subset V \subseteq S$ with $|V| = q+1$. Thus either $f(u, S, V) \cap V \neq \emptyset$ or $f(u, S, V) \cup V \neq S$.
In the first, case let $W = V \cap f(u, S, V)$, and for some ordering label the elements of W: $w_1, ..., w_r$ and the elements of $V \setminus W$: $v_1, ..., v_{q+1-r}$.
If $W = V$, then we construct a communication graph as in figure 2b and consider a broadcast initiated at u. The messages will travel in only one direction through the binary tree and so when w^* receives a message it will be the only message on the graph. This message will then be passed onto the cycle where it will circulate, before being passed back onto the binary tree in the opposite direction. Again the policy of all nodes other than u is indistinguishable from AF and so when u next receives a message, it receives from all of V and these are the only messages (outside of the path from c to d). We therefore have a repeating sequence as u will forward the message back to every node of W.
If $W \subset V$, then we construct a slightly different communication graph (see figure 2c). This graph instead partitions S into W, $V \setminus W$ and $S \setminus V$, with separate binary trees for W and $V \setminus W$. Here we consider a broadcast initiated at a. When u first receives a message it will receive the message from all identifiers in V and there will be no other messages in the body of the graph. Then by assumption u will send a message to some portion of $\{l_1, ..., l_{k-q-1}\}$ as well as all of W. The leaves will not respond and messages will travel only upwards in the binary tree with W as leaves and w^* as its root. Thus, a will next receive a message only from w^* and will send to v^* and c. Until u receives a message the only messages in the body of the graph will be those travelling down the binary tree with $V \setminus W$ as its leaves and they will all arrive at u simultaneously. Thus, u will then receive only from $V \setminus W$. Since $|V \setminus W| < |V| = q + 1$ by assumption u must make the same decision as AF and so will send messages to all of W and its leaves. This creates a repeating sequence and so we have non-termination. Thus, since $f(u, S, V) \cap V \neq \emptyset$ always allows us to construct a communication graph with a non-terminating broadcast, we must have that $f(u, S, V) \cap V = \emptyset$. Now consider the communication graph from figure 2c again but with W an arbitrary strict subset of V. Since $f(u, S, V)$ does not contain any id from V and none of $\{l_1, ..., l_{k-q-1}\}$ will send a message back to u, u sends messages in only a single round. Therefore, u must send to all of $\{l_1, ..., l_{k-q-1}\}$ otherwise some would not receive the message (and so the protocol would not implement broadcast correctly). Thus, $f(u, S, V) = S \setminus V$ and so we have our contradiction.



(a) Two graphs used in the proof of theorem 14 to determine identifiers' response to receiving only a single message. Left: A tree that forbids sending to too small a subset of neighbours. Right: A graph that forbids sending a message to all neighbours.



(b) A graph used in the proof of theorem 14. The graph consists of a star centred at u with its leaves partitioned into two sets of size r and $k-r$. The leaves in the set of size r are connected by a binary tree of depth $\lceil \log_2 r \rceil$ with root w^* to a cycle, which in turn is connected to a path.



(c) A graph used in the proof of Theorem 14. The graph consists of a star centred at u with its leaves partitioned into three sets of size r, $q-r+1$ and $k-q-1$. The leaves in the first two sets are then each joined to a single node labelled by $w*$ and $v*$ respectively by binary trees of depth $\lceil \log_2 q \rceil$. The single nodes are connected to a node labelled a which is the start of a path $ac...d$.

Figure 2 Note that in all figures the path $c...d$ contains all identifiers in $[m]$ not used in labelling the body of the graph, where m is the largest id in the whole labelling.

Therefore, if $f(u, S, T) = S \setminus T$ for $S \subset \mathbb{N}$ where $|S| = k$ and all $T \subseteq S$ such that $0 < |T| \leq q$, then $f(u, S, V) = S \setminus V$ for $V \subset S$ such that $|V| = q + 1$. Thus, by induction since we know this to be true for all u and S when $q = 1$ it must hold for all $q \leq |S|$.

This gives our claim, as for every u and k we can apply this argument and show that for any $k \in \mathbb{N}$, P is AF up to degree k. ◀

B Proof of the Dichotomy (Theorem 15)

We begin by showing that (im)balance is preserved by the operation of AF,

▶ **Lemma 29.** *For any set of initiators $I \subseteq V$ and configuration S on $G = (V, E)$, $A_{I,G}(S)$ is balanced if and only if S is balanced.*

This gives the following immediate corollary, as $\emptyset$ is trivially balanced.

▶ **Corollary 30.** *If S is imbalanced on $G = (V, E)$, then for all $k > 0$, $A^k_G(S) \neq \emptyset$.*

This gives us the forward direction of Theorem 15. For the other direction, we need the notion of message paths and their recurrence.

▶ **Definition 31.** *A message $m = (v_0, v_1)$ in configuration $S \subset V^2$ has a* message path *v_0v_1. We define the rest of its paths recursively, i.e. $m = (v_0, v_1)$ has a message path $v_0...v_{k+1}$ from S on G if:*

- *$v_0v_1...v_k$ is a message path of m in S*
- *The message (v_k, v_{k+1}) exists in $A^k_G(S)$*

We say that a message $m = (v_0, v_1)$ is recurrent *on $G = (V, E)$ from S if m has a message path of the form $v_0v_1...v_0v_1$ on G from S.*

We obtain the following property relating message paths and termination immediately.

▶ **Lemma 32.** *Let S be a non-empty configuration on $G = (V, E)$ such that $A^k_G(S) = S$, then S contains a recurrent message on G.*

The following theorem connects the notions of imbalance and recurrence.

▶ **Theorem 33.** *Let S be a configuration on G, S is imbalanced if and only if it contains a recurrent message.*

Since, all non-terminating configurations must either contain a recurrent message or eventually reach a configuration with a recurrent message, we have that all balanced configurations must terminate. Thus, Theorem 15 follows immediately from Theorem 33. The forward direction is itself immediate from Corollary 30. For the reverse we take $G = (V, E)$ to be a communication graph and $S \subseteq V^2$ to be a balanced configuration of messages. For contradiction we assume that S contains a recurrent message m which has a message path W performing exactly one excursion and return to m. We will view $W = w_0w_1...w_0w_1$ as both a walk on G and a word. The key observation we require is that the number of consecutive steps that can be spent on certain subgraphs by a message path is bounded from a balanced configurations.

▶ **Lemma 34.** *W must obey the following rules:*

1. *W cannot return to the node it just came from, i.e. no sequence of the form uvu.*
2. *W cannot take $2x + 2$ consecutive steps around an odd cycle of length $2x + 1$.*
3. *W cannot take $x + 1$ consecutive steps around an even cycle of length $2x$.*

4. *If W takes more than $x+y+z+2$ consecutive steps on an FEC it remains on one cycle.*
5. *If W takes x steps around an even cycle of length $2x$, then there exists W' which takes the opposite path of equal length and is otherwise identical.*

The fifth of these has the following useful interpretation, if the existence of W' implies imbalance then the existence of W implies imbalance. Therefore, we will use this rule as a substitution allowing us to "modify" W to take the alternate path. By application of these rules we can obtain the following further conditions:

▶ **Lemma 35.** *W must contain the following four rules:*

1. *If W contains a factor $u...u$ then that factor contains an odd cycle as a subfactor.*
2. *There can only be one factor of W that forms an odd cycle.*
3. *Every node appears at most twice in W.*
4. *There exists at most one node that is both a member of a consecutive odd cycle and appears twice. Furthermore, such a node must be the start and end of the cycle.*

No matter how we construct W, it will violate one of these rules.

Proof of the reverse direction of Theorem 33. Since W must have the form $w_0w_1...w_0w_1$, it follows from Lemma 35 (3) that there exists a cycle C containing m such that W fully traverses C before returning to m, with possible excursions. Specifically, there exists a sequence of pairs $(w_0, w_1), (w_1, w_2)...(w_k, w_0), (w_0, w_1)$ such that $C = w_0w_1..w_kw_0$ is a cycle of G, each pair appears in W in consecutive order (i.e. $W = w_0w_1...w_1w_2...w_kw_0..w_0w_1$).

▷ Claim 36. C is an odd cycle

Proof. If C is of even length (say $2k$), then there must be an excursion from C otherwise Lemma 34 (3) would be violated. However, by Lemma 35 (1) and (2), there can be at most one such excursion as it must contain a consecutive odd cycle. Thus, the two subsequences on either side must be factors of W. Therefore, since some subsequence of W traverses C fully with one additional step, one of the two factors must take $k+1$ steps around C. This also violates Lemma 34 (3) and so C must be an odd cycle. ◁

▷ Claim 37. W consists of two odd cycles C and $\hat{C}$ connected by a path.

Proof. If C is an odd cycle there must be an excursion from it or Lemma 34 (2) would be violated. By Lemma 35 (1) and (2) the excursion must contain exactly one consecutive odd cycle which we denote by $\hat{C}$. If $\hat{C}$ does not share its starting node with C, W either forms a path between C and $\hat{C}$ or some chain of cycles. We can use Lemma 34 (5) to eliminate all even cycles of this chain, after which any odd cycles in the chain correspond to a fully traversed FEC when paired with $\hat{C}$ and so violate Lemma 34 (4). Thus, W takes a simple path from C to $\hat{C}$ and back, although possibly intersecting C along the way. ◁

▷ Claim 38. C and $\hat{C}$ intersect with each other but not the path between them.

Proof. If the path to $\hat{C}$ does intersect C, since we are taking the same path in both directions any node shared between the path and C appears in W three times. This violates Lemma 35 (3) and so C must be disjoint from the path to $\hat{C}$. Similarly $\hat{C}$ must be disjoint from the path otherwise it would violate the same lemma. If C is disjoint from $\hat{C}$ the pair would form a fully traversed FEC, thereby violating Lemma 34 (4). Thus, C and $\hat{C}$ must intersect. ◁

▷ Claim 39. Claim: C and $\hat{C}$ do not intersect.

Proof. Let $W = uv..wx_1, ..., x_k w..uv$ where $C = u...w...uv$ and the excursion to $\hat{C}$ is given by $wx_1...x_k w$. Now assume that $\hat{C}$ contains a node from C which occurs strictly before w in W. This node is on a consecutive odd cycle and appears twice. There must exist a latest such node in the ordering of C, we denote it y. Since y is on a consecutive odd cycle and appears twice in W it must be the start and end point of $\hat{C}$ by Lemma 35 (4). However, then since $y \neq w$ it must appear three times, violating Lemma 35 (3). The same argument holds taking the earliest node shared by C and $\hat{C}$ strictly after w. Thus, the only node that can be shared by C and $\hat{C}$ is W, implying that W forms two odd cycles sharing a single node. However, this is an FEC which is fully traversed and so violates Lemma 34 (4). ◁

Thus, W cannot exist and so there can be no recurrent message in S ◀

New Limits on Distributed Quantum Advantage: Dequantizing Linear Programs

Alkida Balliu
Gran Sasso Science Institute, L'Aquila, Italy

Corinna Coupette
Aalto University, Espoo, Finland
Max Planck Institute for Informatics, Saarbrücken, Germany

Antonio Cruciani
Aalto University, Espoo, Finland

Francesco d'Amore
Gran Sasso Science Institute, L'Aquila, Italy

Massimo Equi
Aalto University, Espoo, Finland

Henrik Lievonen
Aalto University, Espoo, Finland

Augusto Modanese
Aalto University, Espoo, Finland

Dennis Olivetti
Gran Sasso Science Institute, L'Aquila, Italy

Jukka Suomela
Aalto University, Espoo, Finland

Abstract

In this work, we give two results that put new limits on distributed quantum advantage in the context of the LOCAL model of distributed computing:

1. We show that there is no distributed quantum advantage for any linear program. Put otherwise, if there is a quantum-LOCAL algorithm $\mathcal{A}$ that finds an α-approximation of some linear optimization problem Π in T communication rounds, we can construct a classical, deterministic LOCAL algorithm $\mathcal{A}'$ that finds an α-approximation of Π in T rounds. As a corollary, all classical lower bounds for linear programs, including the KMW bound, hold verbatim in quantum-LOCAL.
2. Using the above result, we show that there exists a locally checkable labeling problem (LCL) for which quantum-LOCAL is strictly weaker than the classical deterministic SLOCAL model.

Our results extend from quantum-LOCAL to finitely dependent and non-signaling distributions, and one of the corollaries of our work is that the non-signaling model and the SLOCAL model are incomparable in the context of LCL problems: By prior work, there exists an LCL problem for which SLOCAL is strictly weaker than the non-signaling model, and our work provides a separation in the opposite direction.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Quantum computation theory

Keywords and phrases linear programming, distributed quantum advantage, quantum-LOCAL model, SLOCAL model, online-LOCAL model, non-signaling distributions, locally checkable labeling problems, dequantization

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.11

Related Version *Full Version*: `https://arxiv.org/abs/2506.07574`

Funding This work was supported in part by the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI, the European Union - NextGenerationEU under the Italian MUR National Innovation Ecosystem grant ECS00000041 - VITALITY – CUP: D13C21000430001, the PNRR MIUR research project GAMING "Graph Algorithms and MinING for Green agents" (PE0000013, CUP D13C24000430001), and the Research Council of Finland, Grants 363558 and 359104.

Acknowledgements We thank the anonymous reviewers for their helpful comments and suggestions, which allowed us to improve the quality of this paper.



© Alkida Balliu, Corinna Coupette, Antonio Cruciani, Francesco d'Amore, Massimo Equi, Henrik Lievonen, Augusto Modanese, Dennis Olivetti, and Jukka Suomela;
licensed under Creative Commons License CC-BY 4.0



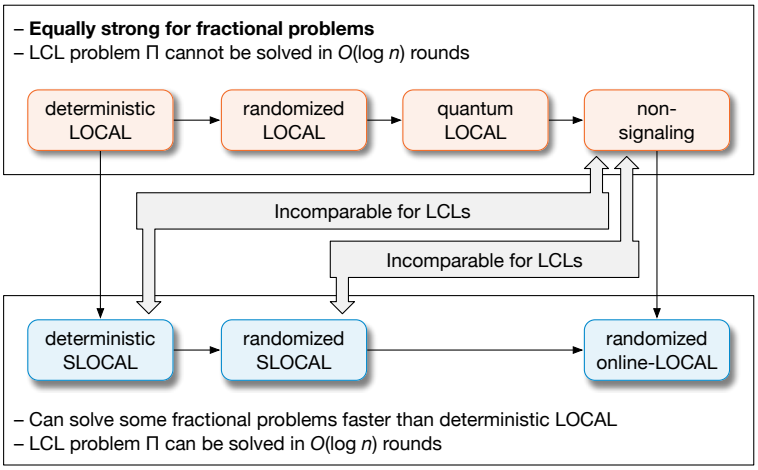


Figure 1 Overview of the results and relevant models of computing.

1 Introduction

In this work, we explore the landscape of distributed graph algorithms in two dimensions:

1. Classical distributed algorithms vs. distributed *quantum* algorithms.
2. Combinatorial graph problems (e.g., maximum independent set) vs. their *fractional* linear-programming relaxations (e.g., maximum *fractional* independent set).

We prove two results that put limits on distributed quantum advantage; see Figure 1 for a schematic overview:

1. We show that there is no distributed quantum advantage for any linear program.
2. Using the above result, we give a new separation between quantum algorithms and classical algorithms (more precisely, between quantum-LOCAL and SLOCAL models).

1.1 Contribution 1: Dequantizing Fractional Algorithms

Setting: Fractional Problems in the LOCAL Model and the Quantum-LOCAL Model. Let us first recall what fractional linear-programming relaxations of graph problems are. For example, in the maximum independent set problem, the task is to label each node $v \in V$ with a value $x_v \in \{0, 1\}$ such that for each edge $\{u, v\}$, we satisfy $x_u + x_v \leq 1$, and we are maximizing $\sum_v x_v$. Now the maximum *fractional* independent set problem is the obvious linear-programming relaxation, where the range of values is $x_v \in [0, 1]$. See Section 2.2 for more details.

In the LOCAL model of distributed computing, a set of computers (nodes) communicates via bidirectional links (edges) defined by an input graph, and computation proceeds in synchronous rounds. In each round, each computer may send a message of unlimited size to each of its neighbors and update its state based on the messages it receives, and the main complexity measure is the number of communication rounds required to solve the given problem. The quantum-LOCAL model is like the LOCAL model, except that we replace all computers with quantum computers and all communication links with quantum communication links capable of exchanging qubits. The main source of potential advantage of the quantum-LOCAL over LOCAL comes from the fact that the messages may contain qubits entangled with the state of the computer, and this has indeed been used to show an advantage for LCL problems [5].

New Result. We prove the following result that is applicable to any fractional linear-programming relaxation:

> Assume $\mathcal{A}$ is a distributed algorithm that finds an α-approximation of some linear optimization problem Π in T communication rounds in the **quantum-LOCAL model**. Then there is also an algorithm $\mathcal{A}'$ that finds an α-approximation of Π in T communication rounds in the **classical deterministic LOCAL model**.

It was already well-known that distributed graph algorithms for solving linear programs can be *derandomized* for free – without losing anything in the running time or approximation ratio. What we show is that they can also be *dequantized*. This can be pushed even further, beyond the quantum-LOCAL model: We show that it holds even if $\mathcal{A}$ is an algorithm in the *non-signaling model*, which is a model strictly stronger than quantum-LOCAL (see Section 2.3 for detailed definitions).

Technical Overview. The proof is near-trivial: $\mathcal{A}'$ outputs the *expected value* of the output of $\mathcal{A}$. A bit more precisely, let X_v be the random variable that represents the output of $\mathcal{A}$ at node v, and let $x_v = \mathbb{E}[X_v]$ be the output produced by $\mathcal{A}'$. Now if we look at the entire output vector, it holds that $x = \mathbb{E}[X]$. In particular, x is a linear combination of α-approximate solutions X to problem Π, and hence x is also an α-approximate solution to Π. We give the details in Section 3.

Implications for the KMW Bound. One of the seminal lower bounds for distributed graph algorithms is the KMW lower bound by Kuhn, Moscibroda, and Wattenhofer [16]; see Appendix C. In 2009, Gavoille, Kosowski, and Markiewicz [14] observed that this lower bound holds also for quantum-LOCAL and non-signaling models. However, to our knowledge, the proof was never published – they merely write that "by careful analysis, it is easy to prove".

However, now we no longer need to do the careful analysis. The key observation is that the KMW bound is inherently a bound for fractional problems. As we now know that there is no quantum advantage for fractional problems, we know that all the implications of the KMW bound indeed hold verbatim also in the quantum-LOCAL model.

While this is not a new result, at least now there is a proof explicitly written down, and the proof is fundamentally different from the idea of carefully inspecting the inner workings of the KMW bound and checking that the argument holds.

1.2 Contribution 2: A New Separation for SLOCAL vs. Non-Signaling Distributions

Setting: LCL Problems and SLOCAL Model. Let us now move on from fractional optimization problems to *locally checkable labeling problems* or LCLs. First defined by Naor and Stockmeyer in the 1990s [17], LCLs constitute a particularly simple family of graph problems that manages to capture many problems of interest in our field. LCL problems are graph problems that can be described by listing a *finite* set of valid labelings in local neighborhoods. There is a long line of work on understanding the landscape of LCL problems, e.g., [1,2,7–10,12,17,18], and this line of research has recently started to explore the interplay

between various models of distributed computing, including not only the classical LOCAL model, quantum-LOCAL model, and non-signaling distributions, but also models such as SLOCAL and online-LOCAL.

The SLOCAL model [15] is a sequential counterpart of the LOCAL model: An adversary queries nodes in a sequential order, and when a node v is queried, the algorithm has to choose the final label of node v. To do that, the algorithm can gather the full information on its radius-T neighborhood, store all this information at node v (for the benefit of other nodes nearby that get queried later), and use all this information to choose its label. We refer to Section 2 for precise definitions, and to Figure 1 for an overview of how SLOCAL is related to other recently-studied models of computing.

New Result. The SLOCAL model is at least as strong as the LOCAL model because with the full knowledge of the radius-T neighborhood, it can simulate any LOCAL algorithm for T steps. However, its exact relation with the quantum-LOCAL model and non-signaling distributions has been an open question – in essence, the question is whether the ability to manipulate qubits is more useful or less useful than the ability to process nodes in some sequential order. We prove the following new result:

> There exists an LCL problem Π such that Π can be solved with locality $O(\log n)$ in the SLOCAL model, but it *cannot* be solved with locality $O(\log^{1.49} n)$ in the non-signaling model or the quantum-LOCAL model.

Technical Overview. While this may seem disconnected from the results in Section 1.1, it is, in a sense, a direct corollary. One implication of the KMW bound is that the problem of finding a maximal matching in a bipartite graph of degree at most Δ cannot be solved in $o(\log \Delta / \log \log \Delta)$ rounds. On the other hand, this is a problem that is trivial to solve in the SLOCAL model in $O(1)$ rounds. So at this point, we have a *family* of LCL problems $\Pi'(\Delta)$ parameterized by Δ that is trivial in SLOCAL but nontrivial in the non-signaling model. We can now plug this family of problems into the construction of [5], as maximal matchings satisfy their key technical requirement of *linearizability*. Deploying this machinery yields a single genuine LCL problem that is strictly easier to solve in SLOCAL than in the non-signaling and quantum-LOCAL models. We give the details in Section 4.

Implications for the Landscape of Models. By recent work [6], there is an LCL problem that is strictly easier to solve in the non-signaling model than in the SLOCAL model. Here, we have obtained a separation in the converse direction. Therefore, in particular:

> The SLOCAL model and the non-signaling model are incomparable in the context of LCL problems; neither is able to simulate the other with constant overhead.

We contrast this with, e.g., the situation for the randomized online-LOCAL model, i.e., the SLOCAL model augmented with a global memory that all nodes can access [2]. The randomized online-LOCAL model *can* simulate both the SLOCAL model and the non-signaling model [1].

2 Preliminaries

Natural numbers are denoted as $\mathbb{N}$ and include 0, while we use $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$ for natural numbers without 0. Moreover, we use the notation $[n] = \{1, \ldots, n\}$ for any $n \in \mathbb{N}_+$. Throughout this work, we consider only undirected graphs. A graph $G = (V, E)$ consists of a set of nodes V and a set of edges E, and we use the notation $V(G)$ and $E(G)$, respectively, if we need to specify which graph we refer to. Given a subset of nodes A, $G[A]$ is the subgraph induced by A, that is, $G[A] = (A, E_A)$, where $(u, v) \in E_A$ if and only if $u, v \in A$ and $(u, v) \in E$.

For any two nodes $u, v \in V$ in graph $G = (V, E)$, $\mathrm{dist}_G(u, v)$ is the length of a shortest path starting from u and ending at v. If there is no ambiguity, we write $\mathrm{dist}(u, v)$ without the graph subscript. For subsets of nodes $A, B \subseteq V$, we can then define $\mathrm{dist}(A, B) = \min_{(u,v)\in A\times B} \mathrm{dist}(u, v)$. For $T \in \mathbb{N}$, we define the radius-T neighborhood of a node u as $\mathcal{N}_T[u] = \{v \mid \mathrm{dist}(u, v) \leq T\}$, i.e., the set of nodes at distance at most T from u. This can be extended to a subset of nodes $A \subseteq V$ as $\mathcal{N}_T[A] = \cup_{u\in A}\mathcal{N}_T[u]$. All the above notions of distances and neighborhoods can be easily generalized to the case where we consider the edges instead of the nodes of a graph.

If v is a node and $e = \{u, v\}$ is an edge, then the pair (v, e) is a *half-edge*, and $\mathcal{H}(G)$ is the set of all pairs (v, e) where $v \in V(G)$ and (v, e) is a half-edge. A *centered graph* is a pair (G, c), where G is a graph and $c \in V(G)$ is one of its nodes, called the *center*. The *eccentricity* of (G, c) is the maximum distance $\max_{u\in V(G)} \mathrm{dist}_G(c, u)$ of a node from the center.

We now introduce the notion of *labeled graph*.

▶ **Definition 1** (Labeled graph [5]). *Suppose that $\mathcal{V}$ and $\mathcal{E}$ are sets of labels. A graph $G = (V, E)$ is said to be $(\mathcal{V}, \mathcal{E})$-*labeled *if the following holds:*

1. *Each node $v \in V$ is assigned a label from $\mathcal{V}$;*
2. *Each half-edge $(v, e) \in V \times E$ satisfying $v \in e$ is assigned a label from $\mathcal{E}$.*

We will consider problems that, in general, take as input a labeled graph and require as output a labeling such that some constraints are satisfied. To formalize the class of problems we consider, for a $(\mathcal{V}, \mathcal{E})$-labeled graph G, let $\ell(v) \in \mathcal{V}$ be the label assigned to node v and $\ell((v, e)) \in \mathcal{E}$ be the label assigned to half-edge (v, e).

To be able to consider labelings restricted to a subset of nodes, we say that, if we are given sets A and B, a subset $A' \subseteq A$, and a function $f : A \to B$, then $f \restriction_{A'}$ is the function $g : A' \to B$ such that $f(x) = g(x)$ for all $x \in A'$, and $\restriction$ is called the restriction operator. Given a $(\mathcal{V}, \mathcal{E})$-labeled graph G with labeling function ℓ, and another $(\mathcal{V}, \mathcal{E})$-labeled graph G' with labeling ℓ', suppose that $\varphi : V(G) \to V(G')$ is an isomorphism between G and G'. We say that ℓ and ℓ' are isomorphic if the labeling is preserved under φ.

2.1 Locally Checkable Labeling (LCL) Problems

In the distributed setting, the well-studied class of *locally checkable labeling (LCL) problems* [17] plays a central role, as they are those problems where the validity of a solution can be checked *locally*, that is, within a constant radius r. The intuition is that a node can gather its radius-r neighborhood to check whether its output satisfies the constraints of the LCL.

To formally define what it means to satisfy an LCL, we first introduce the notion of a set of constraints. This is the set of all valid labelings of a neighborhood of radius r and maximum degree Δ.

▶ **Definition 2** (Set of constraints). *Let $r, \Delta \in \mathbb{N}$ be constants. Consider two finite label sets $\mathcal{V}$ and $\mathcal{E}$. Let $\mathcal{C}$ be a finite set of pairs (H, v_H), where (H, v_H) is a $(\mathcal{V}, \mathcal{E})$-labeled centered graph such that the eccentricity of v_H is at most r and the degree of H is at most Δ. We say that $\mathcal{C}$ is an (r, Δ)-set of constraints over $(\mathcal{V}, \mathcal{E})$.*

After defining the notion of constraint, we can now define the notion of *constraint satisfaction*. Namely, we say that a graph satisfies a set of constraints if all of its radius-r neighborhoods belong to that set.

▶ **Definition 3** (Satisfying a set of constraints). *Let G be a $(\mathcal{V}, \mathcal{E})$-labeled graph, and let $\mathcal{C}$ be an (r, Δ)-set of constraints over $(\mathcal{V}, \mathcal{E})$, for some finite set of labels $\mathcal{V}, \mathcal{E}$. The graph G satisfies $\mathcal{C}$ if the following holds:*

- *For every node $u \in V(G)$, the $(\mathcal{V}, \mathcal{E})$-labeled graph $G[\mathcal{N}_r[u]]$ is such that the centered graph $(G[\mathcal{N}_r[u]], u)$ belongs to $\mathcal{C}$.*

We can now define the notion of *locally checkable labeling (LCL) problems*. An LCL can be seen as a relation that specifies which pairs of input and output labelings are valid.

▶ **Definition 4** (Locally Checkable Labeling (LCL) problems). *Let $r, \Delta \in \mathbb{N}$ be constants, and let $\mathcal{V}_{in}$, $\mathcal{E}_{in}$, $\mathcal{V}_{out}$, and $\mathcal{E}_{out}$ be finite sets of labels. A locally checkable labeling (LCL) problem Π is a tuple $(\mathcal{V}_{in}, \mathcal{E}_{in}, \mathcal{V}_{out}, \mathcal{E}_{out}, \mathcal{C})$ such that the following holds:*

- *$\mathcal{C}$ is an (r, Δ)-set of constraints over $(\mathcal{V}_{in} \times \mathcal{V}_{out}, \mathcal{E}_{in} \times \mathcal{E}_{out})$.*

Suppose that we are given as input a $(\mathcal{V}_{\text{in}}, \mathcal{E}_{\text{in}})$-labeled graph G, and let ℓ_{in} be the input labeling function of G. Solving an LCL problem $\Pi = (\mathcal{V}_{\text{in}}, \mathcal{E}_{\text{in}}, \mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, \mathcal{C})$ on G means to find a labeling function ℓ_{out} that produces an output labeling on G, turning G into a $(\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}})$-labeled graph, such that the following holds:

- For each node $v \in G$, let $\ell(v) = (\ell_{\text{in}}(v), \ell_{\text{out}}(v))$. For each half-edge (v, e) in G, let $\ell((v, e)) = (\ell_{\text{in}}((v, e)), \ell_{\text{out}}((v, e)))$. Then the labeling function ℓ turns G into a $(\mathcal{V}_{\text{in}} \times \mathcal{V}_{\text{out}}, \mathcal{E}_{\text{in}} \times \mathcal{E}_{\text{out}})$-labeled graph that satisfies $\mathcal{C}$ according to Definition 3.

In other words, if we are given a graph and an input labeling, solving an LCL means finding an output labeling under which the graph satisfies the set of constraints of the LCL.

2.2 Distributed Linear Programming (LP) Problems

Next, we define distributed LP problems. We consider the following distributed setting: We are given a communication graph $G = (V, E)$ and a linear program bound to G of the form

$$\begin{aligned} \text{optimize} \quad & \sum_{i \in \mathcal{F}} c_i \cdot x_i \\ \text{subject to} \quad & \sum_{i \in \mathcal{F}} A_{j,i} \cdot x_i \trianglelefteq b_j \quad \forall j \in \mathcal{C} \\ & x_i \geq 0 \quad \forall i \in \mathcal{F} \,, \end{aligned}$$

where $\mathcal{F}$ is the set of variables, $\mathcal{C}$ is the set of constraints, coefficients $A_{j,i}, b_j, c_i$ are known locally, and the inequality $\trianglelefteq$ can be $\leq$, $=$, or $\geq$, depending on the LP formulation. In the distributed setting, each node $v \in V$ in the network "owns" one or more variables $x_i \in \mathcal{F}$. Furthermore, in the LOCAL (resp. SLOCAL) model, each node $v \in V$ knows the local constraints and variables involving nodes within its radius-T neighborhood $\mathcal{N}_T[v]$.

Types of Distributed LPs. There are three classes of distributed LP formulations: node-based, edge-based, and node-edge-based. In the first one, each node $v \in V$ is associated with a variable x_v. In the second, each node has a variable $x_{(v,u)}$ for each $u \in \mathcal{N}[v]$. Since an edge $(u, v) \in E$ is shared by both endpoints, we require that the two local copies coincide, i.e., $x_{(v,u)} = x_{(u,v)}$. In other words, v and u must agree on a common value for their shared edge variable. In the latter formulation, each node is associated with both a variable x_v and a set of variables $x_{(v,u)}$ shared with its neighborhood.

There are different types of local outputs for each class of distributed LP. For node-based LPs, the local output is a real value $x_v \in \mathbb{R}^{\geq 0}$ for each node $v \in V$, whereas for edge-based LPs, each node pair $(v, u) \in E$ agrees on and outputs a real value $x_{(u,v)}$. Consequently, for node-edge LPs, each node outputs both x_v and the incident edge values $x_{(v,u)}$. These local outputs must collectively satisfy the constraint set of the LP. That is, the union of all local outputs across the network must form a globally feasible solution to the LP, meaning that the variable assignments computed and output by the nodes must jointly satisfy all constraints of the LP formulation.

We remark that, in general, distributed LPs are not LCLs. The reason is that LPs involve continuous variables and global feasibility constraints, which cannot be verified using only local information and a finite label set.

Approximation Factor for Distributed LPs. Let $\mathcal{P}$ be a linear program defined over a communication network $G = (V, E)$ with variable set $\mathcal{F}$, and denote the optimal objective value by OPT. Let $\hat{x}$ be a solution produced by a distributed algorithm after a bounded number of synchronous communication rounds. We say that the distributed algorithm achieves an α-*approximation* to $\mathcal{P}$, for some $\alpha \geq 1$, if (1) $\hat{x}$ is a feasible solution to the LP, and (2) $\sum_{i \in \mathcal{F}} c_i \cdot \hat{x}_i \leq \alpha \cdot \text{OPT}$ or $\text{OPT} \leq \alpha \cdot \sum_{i \in \mathcal{F}} c_i \cdot \hat{x}_i$ for minimization or maximization problems, respectively. In other words, the distributed solution is within a factor α of the global optimum, even though each node operates with only local information. If we are dealing with a probabilistic model of computation (e.g., rand-LOCAL or non-signaling; see Section 2.3 below), then we require that the respective algorithm outputs an α-approximation *in expectation*.

Fractional Maximum Matching. In this work, we consider the fractional maximum matching problem formulated as the following LP:

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{(v,u) \in E} x_{(v,u)} \leq 1 \quad \forall v \in V \\ & 0 \leq x_e \leq 1 \quad \forall e \in E \end{aligned}$$

Here, each variable x_e corresponds to an edge $e = (v, u) \in E$ and it is "owned" by both endpoints v and u in G. Each node $v \in V$ is responsible for the constraint $\sum_{u \in \mathcal{N}[v]} x_{(v,u)} \leq 1$, which involves all variables corresponding to the edges incident to v.

Now consider any *maximal matching* in the graph. By definition, a maximal matching is a matching where no additional edge can be added without violating the matching property. It is a standard result in approximation algorithms that any maximal matching is a 2-approximation to the maximum integral matching. Furthermore, the size of a maximum matching can be up to a factor $2/3$ smaller than the fractional maximum matching. Combining these bounds, we obtain that any maximal matching gives a feasible solution to the above LP and a solution value within a factor 3 of the optimum.

2.3 Models

In this section, we define all our computational models of interest.

The LOCAL Model. In the LOCAL model of computing, we are given a distributed system of n processors (or nodes) connected through a communication network represented as a graph $G = (V, E)$, along with an input function x. Every node $v \in V(G)$ has input data $\mathrm{x}(v)$, which encodes the number n of nodes in the network, a unique identifier from the set $[n^c] = \{1, 2, \ldots, n^c\}$, where $c \geq 1$ is a fixed constant, and possible inputs defined by the problem of interest (we assume nodes store both input node labels and input half-edge labels). If computation is randomized, we call the model *randomized LOCAL* (or rand-LOCAL), which means that $\mathrm{x}(v)$ additionally encodes an infinite string of bits that are uniformly and independently sampled for each node, and not shared with the other nodes. If this is not the case and computation is deterministic, we call the model *deterministic LOCAL* (or det-LOCAL). Computation is performed by synchronous rounds of communication. In each round, nodes can exchange messages of unbounded (but finite) size with their neighbors, and then perform an arbitrarily long (but terminating) local computation. Errors occur neither in sending messages nor during local computation. Computation terminates when every node v outputs a label $\ell_{\mathrm{out}}(v)$. The running time of an algorithm is the number of communication rounds, given as a function of n, that are needed to output a labeling that solves the problem of interest. In rand-LOCAL, we also ask that the algorithm solves the problem of interest with probability at least $1 - 1/\operatorname{poly}(n)$, where $\operatorname{poly}(n)$ is any polynomial function in n. If an algorithm runs in T rounds and both communication and computation are unbounded, we can look at it as a function mapping radius-T neighborhoods to output labels in the deterministic case, or to a distribution of output labels in the randomized case. Thus, we say that T is the *locality* of the algorithm.

Depending on the context, we may assume that the computing units are actually the *edges* of the graph, and the local variable $\mathrm{x}(v)$ is stored inside all edges that are incident to v.

The quantum-LOCAL Model. The quantum-LOCAL model is defined like the LOCAL model introduced above, with the following differences. Every processor (node) can locally operate on an unbounded (but finite) number of qubits, applying any unitary transformations, and quantum measurements can be locally performed by nodes at any time. In each communication round, nodes can send an unbounded (but finite) number of qubits to their neighbors. The local output of a node still needs to be an output label encoded in classical bits. As in rand-LOCAL, we ask that an algorithm solves a problem with probability at least $1 - 1/\operatorname{poly}(n)$. A more formal definition of the model can be found in [14].

The SLOCAL Model. The SLOCAL model of computing [15] is a sequential counterpart of the LOCAL model: An algorithm $\mathcal{A}$ processes the nodes sequentially in an order $p = v_1, v_2, \ldots, v_n$. The algorithm must work for any given order p. When processing a node v, the algorithm can query $\mathcal{N}_T[v]$, and $\mathcal{A}$ can read u's state for all nodes $u \in \mathcal{N}_T[v]$. Based on this information, node v updates its own state and computes its output $y(v)$. In doing so, node v can perform unbounded computation, i.e., v's new state can be an arbitrary function of the queried $\mathcal{N}_T[v]$. The output $y(v)$ can be remembered as a part of v's state. The *time complexity* $T_{\mathcal{A},p}(G, \boldsymbol{x})$ of the algorithm on graph G and inputs $\boldsymbol{x} = (x(v_1), x(v_2), \ldots, x(v_n))$ with respect to order p is defined as the maximum T over all nodes v for which the algorithm queries a radius-T neighborhood of v. The *time complexity* $T_{\mathcal{A}}$ of algorithm $\mathcal{A}$ on graph G and inputs $\boldsymbol{x}$ is the maximum $T_{\mathcal{A},p}(G, \boldsymbol{x})$ over all orders p.

The non-signaling Model. The non-signaling model is a model of computing that abstracts from how the actual computation is happening in the network, focusing on a probabilistic description of the valid output labelings. In this model, rather than given algorithms, we are asked to produce *outcomes* (also called strategies) that are functions mapping the input to a probability distribution over output labelings. In other words, given a graph and its input, a probability distribution over output labelings is assigned to the graph such that a sample from the distribution will produce a valid output labeling with high probability. The complexity of the outcome is given by its dependency radius T. This means that an outcome is non-signaling beyond distance T if, for any subset A of the nodes of the graph, modifying the graph or its input at distance greater than T from A does not change the output distribution over A. We proceed to give all the formal details needed to properly define the non-signaling model.

We first formally introduce the concept of *outcome*. For a network G, let x represent the function that maps every node to its input, which includes the input labeling function, port numbers, and unique identifiers. An outcome, then, is a function mapping a network and an input (G, x) to a probability distribution over output labelings.

▶ **Definition 5** (Outcome). *Let $\mathcal{V}, \mathcal{E}$ be sets of labels, and let $\mathcal{F}$ be the family of all input networks (G, x). An outcome O is a function that maps an input network $(G, \mathrm{x}) \in \mathcal{F}$ to a probability distribution $\mathrm{O}(G, \mathrm{x}) = \{(\mathrm{out}_i, p_i)\}_{i \in I}$ defined as follows:*

- *The set I is a set of indices.*
- *The function out_i is a labeling function that maps half-edges and nodes of G to labels in $\mathcal{V}$ and $\mathcal{E}$, respectively, making G a $(\mathcal{V}, \mathcal{E})$-labeled graph.*
- *Each p_i is a non-negative probability and $\sum_{i \in I} p_i = 1$.*

We say that an outcome O *solves* an LCL problem Π over a family of graphs $\mathcal{F}$ with probability $q > 0$ if, for every $G \in \mathcal{F}$ and every input data x, it holds that

$$\sum_{\substack{\mathrm{out}_i \in \mathrm{O}(G,\mathrm{x}): \\ \mathrm{out}_i \text{ solves } \Pi \text{ on } G}} p_i \geq q.$$

Let (G, x) be an input network, and consider any subset of nodes $S \subseteq V(G)$. Let $\mathcal{H}(G)[S]$ be the subset of $\mathcal{H}(G)$ that contains half-edges (v, e) for $v \in S$. The restriction of the output distribution $\mathrm{O}(G, \mathrm{x}) = \{(\mathrm{out}_i, p_i)\}_{i \in I}$ to S is the distribution $\mathrm{O}(G, \mathrm{x})[S] = \{(\mathrm{out}_j, p'_j)\}_{j \in J}$, where the output-labeling functions $\{\mathrm{out}_j\}$ assign labels only on nodes of S and on half-edges of $\mathcal{H}(G)[S]$, and the probability p'_j satisfies the following condition:

$$p'_j = \sum_{\substack{\mathrm{out}_i \in \mathrm{O}(G,\mathrm{x}): \\ \mathrm{out}_i \text{ coincides with } \mathrm{out}_j \\ \text{on } S \text{ and } \mathcal{H}(G)}} p_i.$$

We now define the notion of isomorphic output distributions. Consider two graphs G and G' such that $\varphi : V(G) \to V(G')$ is an isomorphism. A probability distribution $\{(\mathrm{out}_i, p_i)\}_{i \in I}$ over output labelings for G is isomorphic to a probability distribution $\{(\mathrm{out}_j, p'_j)\}_{j \in J}$ over output labelings for G' if they are preserved under the action of φ.

We would now like to define a special type of outcome called *non-signaling outcome*. To this end, we first need to define the concept of *view* up to distance T. Given an input network (G, x) and a subset of its nodes $A \subseteq V(G)$, consider the subgraph $G[\mathcal{N}_T[A]]$ induced by $\mathcal{N}_T[A]$. The view up to distance T of A is the pair $\mathcal{V}_T(A) = (G_A, \mathrm{x}_A)$, where G_A is the graph defined as $V(G_A) = V(G[\mathcal{N}_T[A]]) = \mathcal{N}_T[A]$ and $E(G_A) = \{(u, v) \mid (u, v) \in$

$\mathrm{E}(G[\mathcal{N}_T[A]]), \mathrm{dist}_G(u, A) < T$ or $\mathrm{dist}_G(v, A) < T\}$, and $\mathrm{x}_A = \mathrm{x} \restriction \mathcal{N}_T[A]$. Intuitively, nodes in A see everything up to distance T except for the edges among the bordering nodes of $G[\mathcal{N}_T[A]]$ (but they can see the labels of the half-edges incident to them). In general, for two arbitrary graphs G and H and subsets of nodes $A \subseteq V(G), B \subseteq V(H)$, we say that a function $\varphi : V(G) \to V(H)$ is an isomorphism between $\mathcal{V}_T(A) = (G_A, \mathrm{x}_A)$ and $\mathcal{V}_T(B) = (G_B, \mathrm{x}_B)$ if φ is an isomorphism between $V(G_A)$ and $V(G_B)$ and $\mathrm{x}_A = \mathrm{x}_B \circ \varphi$. Now, a non-signaling outcome is defined as follows.

▶ **Definition 6** (Non-signaling outcome). *Let* O *be an outcome, G and H be graphs, $\varphi : V(G) \to V(H)$ be a function, and $T \in \mathbb{N}$. Outcome* O *is* non-signaling beyond distance T *if, for any two subsets of nodes $A_G \subseteq V(G), A_H \subseteq V(H)$ such that φ is an isomorphism between $\mathcal{V}_T(A_G)$ and $\mathcal{V}_T(A_H)$, the restricted distributions $\mathrm{O}(G, \mathrm{x}_G)[A_G]$ and $\mathrm{O}(H, \mathrm{x}_H)[A_H]$ are isomorphic under φ.*

Alternatively, we can also say that O has locality T. Running T-round classical or quantum-LOCAL algorithms, both with or without shared resources, yields output-labeling distributions that are non-signaling outcomes with locality T.

The *non-signaling model* is, thus, a computational model where the input is a network with input (G, x), and an LCL problem Π is solved if there exists a non-signaling outcome O that solves Π with success probability at least $1 - 1/\operatorname{poly}(n)$, where $n = |V(G)|$.

When the input of a problem is clear from the context, we will omit the input network (G, x), writing $\mathrm{O}(G)$. Observe that all the concepts of views and of restrictions of outcomes can be naturally defined via half-edges instead of nodes, especially when dealing with problems that only ask us to label half-edges (such definitions will be used later in Section 4.5).

We further assume that all probability distributions and output labelings that define a non-signaling O are *computable*. This is because a proper quantum-LOCAL algorithm can be implemented by a quantum circuit, which can be simulated in a classical computer (with costly computation). Hence, its output distribution is computable, and we can restrict ourselves to computable output-labeling distributions.

3 Dequantization for Distributed Linear Programming Problems

In this section, we prove our first result, i.e., that distributed non-signaling (and in particular also quantum-LOCAL) has no advantage over det-LOCAL for distributed linear programming problems.

▶ **Theorem 7.** *Let $\mathcal{P}$ be a distributed linear programming problem that admits a non-signaling distribution over α-approximations with locality T. Then there exists a deterministic LOCAL algorithm that finds an α-approximation of $\mathcal{P}$ with locality T.*

For simplicity, we will consider only the case where $\mathcal{P}$ is a node-based problem. It is clear how to extend the proof to the other classes of distributed LPs. The idea of the proof is relatively simple: We first note that for any distribution of α-approximations of a linear program $\mathcal{P}$, the *expectation* is also an α-approximation; this follows directly from the convexity of a linear program and the linearity of expectation. A detailed proof can be found in Appendix B.

▶ **Lemma 8.** *Let $\mathcal{P}$ be a linear program, and let* O *be a distribution over α-approximations of $\mathcal{P}$. Then $\hat{x} = \mathbb{E}[\mathrm{O}]$ is also an α-approximation of $\mathcal{P}$.*

Then we show that there exists a LOCAL algorithm that can locally compute this expectation, given access to the distribution. Intuitively, the algorithm gathers its radius-T neighborhood, where T is the locality of the non-signaling distribution, then calls the outcome

function O on its neighborhood to obtain a distribution of output labelings, and finally outputs the expected value of such a distribution. Again, the technical details are deferred to Appendix B.

▶ **Lemma 9.** *Let* O *be a computable non-signaling distribution with locality* T *over graph family* $\mathcal{F}$*. Then there exists a LOCAL algorithm with locality* T *that computes the expected outcome of this distribution everywhere.*

We are now ready to state the proof of Theorem 7.

Proof of Theorem 7. Let $\mathcal{P}$ be a distributed linear programming problem and let O be computable non-signaling distribution with locality T over α-approximations of $\mathcal{P}$. By Lemma 9, we have a LOCAL algorithm $\mathcal{A}$ that computes the expectation of O locally everywhere. As the outcome is a vector over local elements, its expectation is a vector over the expectations of the local elements. Hence, $\mathcal{A}$ computes the expectation of O. By Lemma 8, this is an α-approximation of $\mathcal{P}$. ◀

4 Separation Between SLOCAL and non-signaling

In this section, we prove that there exists an LCL problem Π that has complexity $O(\log n)$ in the SLOCAL model and complexity $\omega(\log n)$ in the non-signaling model.

▶ **Theorem 10.** *There exists an LCL problem* Π *that has complexity* $O(\log n)$ *in the deterministic SLOCAL model and* $\Omega\left(\log n \cdot \sqrt{\frac{\log n}{\log\log n}}\right)$ *in the non-signaling model.*

We devote the rest of this section to proving Theorem 10.

4.1 Overview

In order to define the problem Π, we borrow ideas from [5]. We start by giving a recap of the main ideas presented in [5].

Recap of Previous Results. The authors of [5] introduced the notion of *linearizable problems*, which are locally checkable problems that are not necessarily LCLs. They proved that, if there exists some linearizable problem P with some complexity $f(n)$ for some function f, then there exists some LCL problem $\Pi = \text{lift}(P)$ with some complexity $f'(n)$, where f' depends on f. For small-enough f, the function f' is a multiplicative factor $\Theta(\log n)$ larger than f. Interestingly, both the quantum complexity and the standard complexity are increased by this $\Theta(\log n)$ factor. In more detail, the authors of [5] proved the following:

1. In [3], it has been shown that there exists a problem P with quantum complexity $O(1)$ and randomized LOCAL complexity $\Omega(\min\{\Delta, \log_\Delta \log n\})$. By taking a suitable value of Δ, this result implies a lower bound of $\Omega(\frac{\log\log n}{\log\log\log n})$.
2. The problem P can be expressed as a linearizable problem.
3. The authors defined a function lift that takes as input a linearizable problem P and returns an LCL problem $\Pi = \text{lift}(P)$.
4. The authors showed that $\Pi = \text{lift}(P)$ has the following complexities:
   - $O(\log n)$ in quantum-LOCAL.
   - $\Omega(\log n \cdot \frac{\log\log n}{\log\log\log n})$ in rand-LOCAL.

Note that, while the problem P itself does not have any super-constant lower bound when $\Delta = O(1)$, this construction allows us to nevertheless obtain a problem $\Pi = \text{lift}(P)$ with a super-constant lower bound as a function of n on graphs in which $\Delta = O(1)$.

Our Approach. In essence, we show that the construction of [5] also preserves complexities in SLOCAL and in non-signaling, and we will use the maximal matching problem, phrased as a linearizable problem, as P. We will obtain the following:

1. Our problem P will be the maximal matching problem, phrased as a linearizable one.
2. The problem P has complexity $O(1)$ in SLOCAL, even on graphs of unbounded degree.
3. The problem $\Pi = \text{lift}(P)$ has complexity $O(\log n)$ in SLOCAL.
4. Maximal matching has a rand-LOCAL lower bound of $\Omega(\sqrt{\frac{\log n}{\log\log n}})$, proved as part of the KMW lower bound [16].
5. Maximal matching is a 3-approximation of fractional maximum matching, and the KMW bound, which is more general, holds for this latter problem as well. Thus, by Theorem 7, the lower bound for maximal matching also holds in non-signaling.
6. We prove that, in non-signaling, an upper bound of $o(\log n \cdot \sqrt{\frac{\log n}{\log\log n}})$ for Π would imply an upper bound of $o(\sqrt{\frac{\log n}{\log\log n}})$ for P, contradicting the KMW lower bound.

In order to achieve this, we prove a result similar to the one of [5], with the difference that we consider SLOCAL upper bounds, and non-signaling lower bounds.

4.2 The Definition of $\Pi = \text{lift}(P)$

We summarize the definition of $\Pi = \text{lift}(P)$ that appeared in [5]. At a high level, the problem Π is an LCL with inputs (i.e., nodes and node-edge pairs have input labels that come from a finite set) that is defined as a combination of two LCL problems:

- The LCL problem Π^{badGraph}, which is a problem defined on any graph.
- The LCL problem Π^{promise}, which is a problem defined on some specific class $\mathcal{G}$ of graphs labeled with some input (which is the input of Π). Note that, in order for a graph G to be in $\mathcal{G}$, the input given to the nodes of G must satisfy some specific local constraints.

In particular, Π^{badGraph} asks us to produce some output that satisfies, among others, the following properties:

- Each node is either marked (labeled with some specific output labels) or unmarked (labeled $\perp$).
- If $G \in \mathcal{G}$, then no node of G is marked.

Moreover, it is shown that there exists a deterministic $O(\log n)$ LOCAL algorithm $\mathcal{A}$ solving Π^{badGraph} on any graph G, such that the output of $\mathcal{A}$ satisfies that each connected component induced by unmarked nodes is in $\mathcal{G}$. Specifically, the authors of [5] proved the following.

▶ **Lemma 11** ([5]). *Let $G \in \mathcal{G}$. Then, the only valid solution for Π^{badGraph} on G is the one assigning $\perp$ to all nodes.*

▶ **Lemma 12** ([5]). *Let G be any graph. There exists a solution for Π^{badGraph} where each connected component induced by nodes outputting $\perp$ is a graph in $\mathcal{G}$. Moreover, such a solution can be computed in $O(\log n)$ deterministic rounds in the LOCAL model.*

The problem Π is defined such that it is first required to solve Π^{badGraph}, and then, on each connected component induced by unmarked nodes, it is required to solve Π^{promise}. We will later describe the problem Π^{promise}, the definition of which will depend on the given linearizable problem P. Now we argue that, in order to prove our lower and upper bounds, we can restrict our attention to graphs that are in $\mathcal{G}$ and to the problem Π^{promise}.

In [5], the quantum-LOCAL upper bound for Π is obtained as follows:

- First, apply Lemma 12. That is, in $O(\log n)$ deterministic LOCAL rounds, we obtain a solution for Π^{badGraph} satisfying that each connected component induced by nodes outputting $\perp$ is a graph in $\mathcal{G}$.

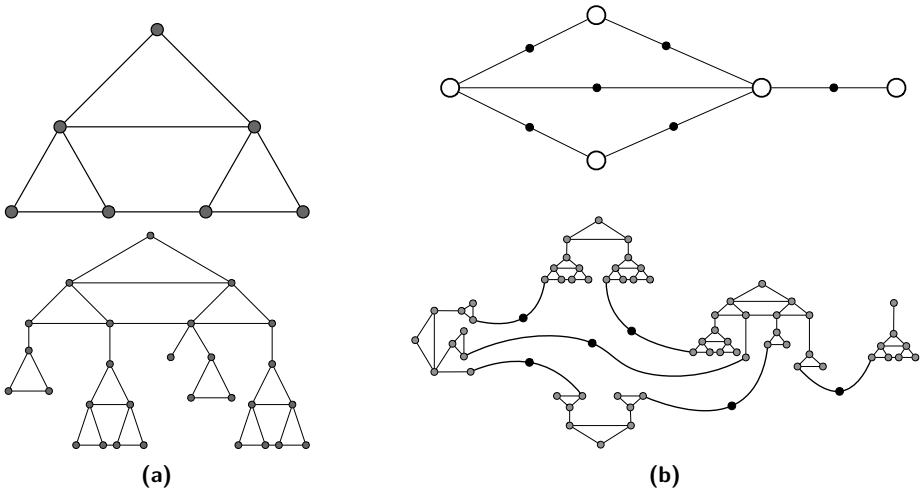


Figure 2 **(a)** A tree-like gadget at the top, and an octopus gadget at the bottom. **(b)** A proper instance at the bottom and, at the top, the graph obtained by contracting each octopus gadget into a single node.

- Then, use a quantum algorithm to solve Π^{promise} on each connected component induced by nodes outputting $\perp$. This requires $O(\log n)$ quantum rounds.

Since an $O(\log n)$ deterministic LOCAL algorithm can be directly executed in SLOCAL (i.e., SLOCAL is at least as strong as LOCAL), and since an SLOCAL algorithm obtained by composing two different SLOCAL algorithms has an asymptotic complexity equal to the sum of the complexities of the two composed algorithms, it is clear from the quantum algorithm that, in order to provide an $O(\log n)$ SLOCAL algorithm for Π, it is sufficient to provide an $O(\log n)$ SLOCAL algorithm for Π^{promise} on graphs that are in $\mathcal{G}$.

The randomized LOCAL lower bound for Π is obtained by considering graphs $G \in \mathcal{G}$. On these graphs, by Lemma 11, the only valid solution for Π^{badGraph} is the one assigning $\perp$ to all nodes. By the definition of Π, this implies that, on G, it is required to solve Π^{promise}. For our non-signaling lower bound, we will follow the exact same strategy.

4.3 The Graph Family $\mathcal{G}$

In order to define the family $\mathcal{G}$, we need to first introduce the notion of *proper instances*. At a high level, a proper instance is a graph that can be obtained by starting from some graph G' (which is not necessarily a simple graph) and replacing nodes with some *gadgets* according to some rules. Then, a graph $G \in \mathcal{G}$ will be obtained by labeling a proper instance in some specific way. In the following, we report the definition of some objects as given in [5]. The basic building block is the notion of *tree-like gadget*, of which we give an example in Figure 2a (top), while the formal definition can be found in Appendix A (Definition 23).

The next building block is called *octopus gadget*. At a high level, an octopus gadget is obtained by starting from a tree-like gadget, and connecting one or two additional tree-like gadgets to each "leaf" of the tree-like gadget. See Figure 2a (bottom) for an example and Appendix A (Definition 24) for a formal definition.

We can now define the family of proper instances. An example is shown in Figure 2b.

▶ **Definition 13** (Proper instance [5]). *Let $G = (V, E)$ be a graph. We say that G is a* proper instance *if there exists a node labeling function $\lambda : V \to \{\mathsf{intra\text{-}octopus}, \mathsf{inter\text{-}octopus}\}$ with the following properties.*

1. *Every connected component in the subgraph induced by nodes labeled* intra-octopus *is an octopus gadget (according to Definition 24).*
2. *The subgraph induced by nodes labeled* inter-octopus *does not contain any edge.*
3. *A node v labeled* intra-octopus *is connected to a node labeled* inter-octopus *if and only if v has coordinates $(w-1, 0)$ in the port gadget P containing v, where w is the height of P (that is, v is the left-most leaf of the port gadget containing v).*

The authors of [5] proved that proper instances are *locally checkable*, in the sense that there exist a set of local constraints $\mathcal{C}$ and finite sets of labels $\mathcal{V}$ and $\mathcal{E}$ for which an arbitrary graph G can be $(\mathcal{V}, \mathcal{E})$-labeled such that the constraints $\mathcal{C}$ to be satisfied on all nodes, if and only if G is a proper instance. More precisely, they proved the following statements.

▶ **Lemma 14** ([5]). *Let G be any non-empty connected graph that is $(\mathcal{V}, \mathcal{E})$-labeled such that $\mathcal{C}$ is satisfied at all nodes. Then G is a proper instance according to Definition 13.*

▶ **Lemma 15** ([5]). *Let G be a proper instance as defined in Definition 13. Then there exists a $(\mathcal{V}, \mathcal{E})$-labeling of G that satisfies the constraints in $\mathcal{C}$ at all nodes.*

We are now ready to define the family $\mathcal{G}$.

▶ **Definition 16.** *A $(\mathcal{V}, \mathcal{E})$-labeled graph G is in $\mathcal{G}$ if and only if the constraints in $\mathcal{C}$ are satisfied at all nodes.*

4.4 Linearizable Problems

In order to define Π^{promise}, we first need to introduce the notion of *linearizable problems*. A linearizable problem $\Pi^{\mathsf{linearizable}} = (\Sigma, (F, L, P), B)$ is defined as follows. Let us consider a hypergraph described as its bipartite incidence graph, where white nodes represent the original nodes, black nodes represent hyperedges, and we are also given an ordering of the edges of the white nodes. Intuitively, a problem $\Pi^{\mathsf{linearizable}} = (\Sigma, (F, L, P), B)$ is linearizable if it is possible to encode it as another LCL where the constraints for white nodes can be expressed solely in terms of consecutive edges in the ordering. More specifically, sets F and L define which labels are allowed for the first and last edges, respectively, while set P contains pairs of labels that can appear consecutively. In other words, P specifies the combinations in which labels can be assigned to a node and its successor in the ordering. We refer to Appendix A (Definition 25) for a formal definition.

We will use *maximal matching* as the running example of a linearizable problem. This is an important step to reach our goal, as we will use the fact that maximal matching can be expressed as a linearizable problem to separate SLOCAL from non-signaling. Maximal matching is a problem defined on graphs, and hence, we will describe a linearizable problem on hypergraphs of rank 2. In this case, the black constraint describes the edge constraints, and the white constraint describes the node constraints. An example of a solution to the maximal matching problem encoded as a linearizable problem is provided in Figure 3, and the following lemma formally states the existence of a linearized version of maximal matching. We defer the details of the proof to Appendix B.

▶ **Lemma 17.** *There exists a linearizable problem $\Pi^{\mathsf{linearizable}} = (\Sigma, (F, L, P), B)$ satisfying the following:*

- *A solution for $\Pi^{\mathsf{linearizable}}$ can be converted into a maximal matching in 0 deterministic LOCAL rounds.*
- *A solution for maximal matching can be converted into a solution for $\Pi^{\mathsf{linearizable}}$ in 0 deterministic LOCAL rounds.*

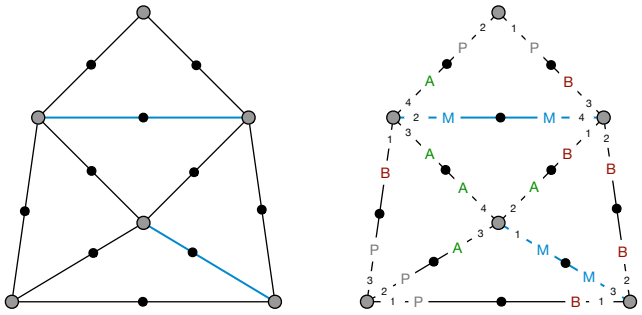


Figure 3 On the left, we show a solution to the maximal matching problem, where black nodes represent hyperedges of rank 2 and blue edges are in the matching. On the right, the same solution is encoded as a solution to $\Pi^{\text{linearizable}}$.

We now connect the notion of a linearizable problem with the problem Π^{promise} that we aim to define. The problem Π^{promise} is defined in [5] as an LCL problem (i.e., by describing its local constraints). This problem is defined as a function of a given problem $\Pi^{\text{linearizable}} = (\Sigma, (F, L, P), B)$. For our purposes, we do not need the details of the definition of Π^{promise}, and it is sufficient to state the properties that any valid solution needs to satisfy. Informally, Π^{promise} is defined such that, if we contract each octopus gadget into a single white node, and we treat each inter-cluster node as a black node, it must hold that, in the resulting graph, we get a solution for $\Pi^{\text{linearizable}}$. The formal definition of Π^{promise} can be found in Appendix A (Definition 26) or in [5].

4.5 SLOCAL Upper Bound and non-signaling Lower Bound

We now establish both an upper bound in the SLOCAL model and a lower bound in the non-signaling model, deferring the formal proofs to Appendix B. For SLOCAL, we have the following upper bound.

▶ **Lemma 18.** *Let $T(n)$ be an upper bound on the SLOCAL complexity of $\Pi^{\text{linearizable}}$ that holds also if the given graph contains parallel edges. Then the SLOCAL complexity of Π is upper bounded by $O(T(n)\log n)$.*

For non-signaling, we have the following lower bound.

▶ **Lemma 19.** *Let $T(n)$ be a lower bound on the locality of $\Pi^{\text{linearizable}}$ in non-signaling with failure probability $p(n)$, which is a non-increasing function of n bounded above by some constant $q < 1$. Then any non-signaling outcome for Π^{promise} with failure probability at most $p(n)$ requires locality $\Omega(T(n^{1/3})\log n)$.*

The following lemma is the key ingredient for the proof of Lemma 19.

▶ **Lemma 20.** *Let $\Pi^{\text{linearizable}}$ be any linearizable problem, and consider the LCL problem $\Pi = \text{lift}(\Pi^{\text{linearizable}})$. Suppose that there exists an outcome O that is non-signaling beyond distance $T(n)$ that solves Π with failure probability $p(n)$, which is a non-increasing function of n bounded above by some constant $q < 1$. Then we can construct an outcome O' that solves $\Pi^{\text{linearizable}}$ with failure probability at most $p(n)$ and is non-signaling beyond distance $T'(n) = O(T(n^3)/\log n)$.*

The proof of Lemma 19 is now straightforward.

Proof of Lemma 19. If any non-signaling outcome for Π with failure probability $p'(n) \leq p(n)$ has locality $o(T(n^{1/3}) \cdot \log n)$, then we can construct a non-signaling outcome that solves $\Pi^{\mathsf{linearizable}}$ with failure probability at most $p'(n)$ and has locality $o(T(n))$ by Lemma 20, which is a contradiction with the fact that we assumed $T(n)$ to be a lower bound on the locality of $\Pi^{\mathsf{linearizable}}$. ◀

4.6 Instantiating Our Construction

We are now ready to prove Theorem 10, referring to Appendix C for a high-level description of the KMW lower bound and to [11] for a detailed exposition. We consider the maximal matching problem, and by Lemma 17, there exists a linearizable problem P that is equivalent to maximal matching. In SLOCAL, maximal matching, and hence P, can be solved in one deterministic round by a trivial greedy algorithm. Hence, by Lemma 18, the deterministic SLOCAL complexity of $\Pi = \mathrm{lift}(P)$ is $O(\log n)$.

For a lower bound, recall that KMW gives us a lower bound of $\Omega(\sqrt{\log n / \log\log n})$ for $O(\log \Delta)$-approximation of fractional maximum matching (see Theorem 27) on graphs of degree $\Delta = 2^{\Theta(\sqrt{\log n \log\log n})}$. Combining Theorem 7 with the KMW lower bound and the fact that fractional maximum matching can be expressed as a linear program, we obtain the following corollary.

▶ **Corollary 21.** *There does not exist a non-signaling distribution for $O(\log \Delta)$-approximation of fractional maximum matching with locality $o(\sqrt{\log n / \log\log n})$ on graphs of degree $\Delta = 2^{\Theta(\sqrt{\log n \log\log n})}$.*

Since a maximal matching is a 2-approximation of a maximum matching, and a maximum matching is a $\frac{3}{2}$-approximation of a maximum fractional matching, we obtain the following.

▶ **Corollary 22.** *There does not exist a non-signaling distribution for maximal matching with locality $o(\sqrt{\log n / \log\log n})$.*

By combining Corollary 22 with Lemma 19, we obtain that in non-signaling, Π has complexity $\Omega(\log n \cdot \sqrt{\frac{\log n}{\log\log n}})$, as desired.

References

1 Amirreza Akbari, Xavier Coiteux-Roy, Francesco D'Amore, François Le Gall, Henrik Lievonen, Darya Melnyk, Augusto Modanese, Shreyas Pai, Marc-Olivier Renou, Václav Rozhon, and Jukka Suomela. Online locality meets distributed quantum computing. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 1295–1306. ACM, 2025. doi:10.1145/3717823.3718211.

2 Amirreza Akbari, Navid Eslami, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. Locality in online, dynamic, sequential, and distributed graph algorithms. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.10.

3 Alkida Balliu, Sebastian Brandt, Xavier Coiteux-Roy, Francesco D'Amore, Massimo Equi, François Le Gall, Henrik Lievonen, Augusto Modanese, Dennis Olivetti, Marc-Olivier Renou, Jukka Suomela, Lucas Tendick, and Isadora Veeren. Distributed quantum advantage for local problems. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM*

Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025, pages 451–462. ACM, 2025. doi:10.1145/3717823.3718233.

4 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 299–308. ACM, 2020. doi:10.1145/3382734.3405715.

5 Alkida Balliu, Filippo Casagrande, Francesco D'Amore, Massimo Equi, Barbara Keller, Henrik Lievonen, Dennis Olivetti, Gustav Schmid, and Jukka Suomela. Distributed quantum advantage in locally checkable labeling problems. *CoRR*, abs/2504.05191, 2025. doi:10.48550/arXiv.2504.05191.

6 Alkida Balliu, Mohsen Ghaffari, Fabian Kuhn, Augusto Modanese, Dennis Olivetti, Mikaël Rabie, Jukka Suomela, and Jara Uitto. Shared randomness helps with local distributed problems. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICALP.2025.16.

7 Sebastian Brandt. An automatic speedup theorem for distributed problems. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 379–388. ACM, 2019. doi:10.1145/3293611.3331611.

8 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM Journal on Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

9 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM Journal on Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

10 Xavier Coiteux-Roy, Francesco D'Amore, Rishikesh Gajjala, Fabian Kuhn, François Le Gall, Henrik Lievonen, Augusto Modanese, Marc-Olivier Renou, Gustav Schmid, and Jukka Suomela. No distributed quantum advantage for approximate graph coloring. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1901–1910. ACM, 2024. doi:10.1145/3618260.3649679.

11 Corinna Coupette and Christoph Lenzen. A breezing proof of the KMW bound. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 184–195. SIAM, 2021. doi:10.1137/1.9781611976496.21.

12 Sameep Dahal, Francesco D'Amore, Henrik Lievonen, Timothé Picavet, and Jukka Suomela. Brief announcement: Distributed derandomization revisited. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 40:1–40:5. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.40.

13 Kőnig Dénes. Graphok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és természettudományi értes itő*, 34:104–119, 1916.

14 Cyril Gavoille, Adrian Kosowski, and Marcin Markiewicz. What can be observed locally? In Idit Keidar, editor, *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings*, volume 5805 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2009. doi:10.1007/978-3-642-04355-0_26.

15 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 784–797. ACM, 2017. doi:10.1145/3055399.3055471.

16 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 63(2):17:1–17:44, 2016. doi:10.1145/2742012.

17 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

18 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020. doi:10.1145/3357713.3384298.

A Omitted Definitions

▶ **Definition 23** (Tree-like gadget [4, 5]). *A graph G is a* tree-like *gadget of height ℓ if it is possible to assign coordinates (l_u, k_u) to each node $u \in G$, where*

- *$0 \leq l_u < \ell$ denotes the depth of u in the tree, and*
- *$0 \leq k_u < 2^{l_u}$ denotes the position of u (according to some order) in layer l_u,*

such that there is an edge connecting two nodes $u, v \in G$ with coordinates (l_u, k_u) and (l_v, k_v) if and only if:

- *$l_u = l_v$ and $|k_u - k_v| = 1$, or*
- *$l_v = l_u - 1$ and $k_v = \lfloor \frac{k_u}{2} \rfloor$, or*
- *$l_u = l_v - 1$ and $k_u = \lfloor \frac{k_v}{2} \rfloor$.*

▶ **Definition 24** (Octopus gadget [5]). *Let $x \geq 1$ be a natural number, and $\eta = (\eta_0, \ldots, \eta_{2^{x-1}-1})$ a vector of 2^{x-1} entries in $\{1, 2\}$. Let $W = \{w_{(i,j)}\}_{(i,j)\in I}$ be a family of positive integer weights, where I is the set containing all pairs (i, j) satisfying $(i, j) \in \{0, 1, \ldots, 2^{x-1} - 1\} \times \{1, 2\}$ and $j \leq \eta_i$.*

*A graph $G = (V, E)$ is an (x, η, W)-*octopus gadget *if there exists a labeling $\lambda : V \to \mathcal{L} = I \cup \{root\}$ of the nodes of G such that the following holds.*

1. *For each element $y \in \mathcal{L}$, let G_y be the subgraph of G induced by nodes labeled with y. Then, for all $y \in \mathcal{L}$, G_y must be a tree-like gadget according to Definition 23.*
2. *For all $y, z \in \mathcal{L}$ such that $y \neq z$, G_y and G_z must be disjoint.*
3. *G_{root} has height x and, for all $(i, j) \in I$, $G_{(i,j)}$ has height $w_{(i,j)} \in W$.*
4. *For all $(i, j) \in I$, there is an edge connecting the node of $G_{(i,j)}$ that has coordinates $(0, 0)$ with the node of G_{root} that has coordinates $(x - 1, i)$.*

G_{root} is called the head-gadget *and, for all $(i, j) \in I$, $G_{(i,j)}$ is called a* port gadget*.*

▶ **Definition 25** (Linearizable problem [5]). *Let H be a hypergraph, and let G be its bipartite incidence graph. Let the nodes of G corresponding to the nodes of H be called* white *nodes, and let the nodes of G corresponding to the hyperedges of H be called* black *nodes.*

- *The task requires to label each edge of G with a label from some finite set Σ.*
- *There is a list of allowed* black node configurations *B, which is a list of multisets of labels from Σ that describes valid labelings of edges incident on a black node. We say that a black node* satisfies the black constraint *if the multiset of labels assigned to its incident edges is in B. It is assumed that the rank of H, and hence the maximum degree of black nodes, is a constant.*
- *Constraints on white nodes are described as a triple (F, L, P), where F (which stands for* first*) and L (which stands for* last*) are finite sets of labels, and P (which stands for* pairs*) is a finite set of ordered pairs of labels. In this formalism, it is assumed that an ordering on the incident edges of a white node is given, and it is required that:*
  - *The first edge is labeled with a label from F;*
  - *The last edge is labeled with a label from L;*

- *Each pair of consecutive edges must be labeled with a pair of labels from P.*

We say that a white node satisfies the white constraint *if its incident edges are labeled in a valid way.*

When solving a linearizable problem in the distributed setting, it is assumed that each node knows whether it is white or black.

▶ **Definition 26** (The definition of Π^{promise} of [5], rephrased)**.** *Given a graph $G \in \mathcal{G}$, the problem Π^{promise} requires to label the nodes of the graph as follows:*

- *Each node that is not in a port gadget must be labeled $\perp$.*
- *Each node that is in a port gadget must be labeled with a label from Σ.*
- *Nodes that belong to the same port gadget must be assigned the same label.*
- *Let g be an octopus gadget, and let $\ell_1, \ldots, \ell_d$ be the labels assigned to the d port gadgets of g, according to the natural left-to-right order of the port gadgets of g. It must hold that $\ell_1 \in F$, $\ell_d \in L$, and $(\ell_i, \ell_{i+1}) \in P$ for all i.*
- *Let v be an inter-cluster node, and let $\ell_1, \ldots, \ell_r$ be the labels assigned to the nodes of its r incident port gadgets. It must hold that $\{\ell_1, \ldots, \ell_r\} \in B$.*

B Omitted Proofs

Proof of Lemma 8. To show that $\hat{x}$ is an approximation of $\mathcal{P}$, we need to establish (1) that $\hat{x}$ is feasible and (2) that $\hat{x}$ gives the correct approximation ratio. To see the feasibility of $\hat{x}$, observe that the feasibility constraints are of the form

$$\sum_{i \in \mathcal{F}} A_{j,i} \cdot x_i \trianglelefteq b_j \quad \forall j \in \mathcal{C} \ .$$

Plugging in $\hat{x}$ and fixing $j \in \mathcal{C}$ gives us

$$\sum_{i \in \mathcal{F}} A_{j,i} \cdot \hat{x}_i = \sum_{i \in \mathcal{F}} A_{j,i} \cdot \mathbb{E}\left[O_i\right] = \mathbb{E}\Big[\underbrace{\sum_{i \in \mathcal{F}} A_{j,i} \cdot O_i}_{\trianglelefteq b_j}\Big] \trianglelefteq b_j \ .$$

The first equality holds by definition, the second equality holds by linearity of expectation, and the conclusion holds by the fact that O is a distribution over feasible solutions and the monotonicity of expectation. Hence, $\hat{x}$ is a feasible solution for $\mathcal{P}$.

It is left to show that $\hat{x}$ is also an α-approximation. Again, we can plug $\hat{x}$ into the target function, obtaining

$$\sum_{i \in \mathcal{F}} c_i \cdot \hat{x}_i = \sum_{i \in \mathcal{F}} c_i \cdot \mathbb{E}\left[O_i\right] = \mathbb{E}\Big[\sum_{i \in \mathcal{F}} c_i \cdot O_i\Big].$$

As each outcome of O is an α-approximation, we can invoke the linearity and monotonicity of expectation and get that this new target is also an α-approximation of $\mathcal{P}$. ◀

Proof of Lemma 9. We give the description for the LOCAL algorithm $\mathcal{A}$: Node v gathers its radius-T neighborhood $\mathcal{N}_T[v]$; this can be done with locality T. It then constructs an arbitrary graph $G' \in \mathcal{F}$ such that the neighborhood of node $v' \in V(G')$ is isomorphic to $\mathcal{N}_T[v]$. Note that such a graph always exists, as the graph the algorithm is being run on is one such graph. Now v invokes the distribution O on graph G' to compute the distribution of outputs for node v' and, in particular, the outcome. Node v then outputs this expected outcome and halts.

It remains to argue that this algorithm computes the expected outcome of O everywhere. This follows directly from the definition of the non-signaling distributions as the marginals of nodes v and v' coincide, and hence their expectations must also coincide. Moreover, the choice of graph G' does not affect this marginal distribution. ◀

Proof of Lemma 17. We define the problem as follows. The label set Σ is defined as $\Sigma = \{\mathsf{M}, \mathsf{B}, \mathsf{A}, \mathsf{P}\}$, where M, B, A, and P stand for *matched*, *before*, *after*, and *pointer*, respectively. The list of allowed black configurations is defined as

$B = \{\{\mathsf{M}, \mathsf{M}\}, \{\mathsf{P}, \mathsf{B}\}, \{\mathsf{P}, \mathsf{A}\}, \{\mathsf{B}, \mathsf{B}\}, \{\mathsf{B}, \mathsf{A}\}, \{\mathsf{A}, \mathsf{A}\}\}$. Then, F is defined as $F = \{\mathsf{M}, \mathsf{B}, \mathsf{P}\}$, L is defined as $L = \{\mathsf{M}, \mathsf{A}, \mathsf{P}\}$, and P is defined as $P = \{(\mathsf{B}, \mathsf{B}), (\mathsf{B}, \mathsf{M}), (\mathsf{M}, \mathsf{A}), (\mathsf{A}, \mathsf{A}), (\mathsf{P}, \mathsf{P})\}$.

We observe that the defined problem $\Pi^{\mathsf{linearizable}}$ satisfies the following properties.

- In any valid solution, for each node v, if we treat the labels assigned to the half-edges incident to v as a string (according to the ordering assigned to the half-edges incident to v), we obtain that such a string must satisfy the regular expression $\mathsf{P}^* \mid \mathsf{B}^*\mathsf{M}\mathsf{A}^*$. We call nodes satisfying P^* *unmatched* nodes and nodes satisfying $\mathsf{B}^*\mathsf{M}\mathsf{A}^*$ *matched* nodes. Moreover, we call an edge *matched* if both its half-edges are labeled M.
- Since $\{\mathsf{P}, \mathsf{P}\} \notin B$, we get that, in any valid solution, unmatched nodes cannot be neighbors.
- Since the label M appears only in the pair $\{\mathsf{M}, \mathsf{M}\}$, and since each matched node must have exactly one incident matched edge, matched edges form an independent set.

This implies that a solution for $\Pi^{\mathsf{linearizable}}$ can be converted into a maximal matching in 0 deterministic LOCAL rounds.

We now show that a maximal matching can be converted into a solution for $\Pi^{\mathsf{linearizable}}$ in 0 deterministic LOCAL rounds. A solution for $\Pi^{\mathsf{linearizable}}$ can be computed as follows:

- Each unmatched node labels P all its incident half-edges.
- Each matched node labels M its incident half-edge e that is part of the matching, B all edges that come before e in the given ordering, and A all edges that come after e in the given ordering.

It is easy to see that the computed solution satisfies the constraints of $\Pi^{\mathsf{linearizable}}$. ◀

Proof of Lemma 18. Let $\mathcal{A}$ be an SLOCAL algorithm for $\Pi^{\mathsf{linearizable}}$ with complexity $T(n)$. We show how to use $\mathcal{A}$ to solve Π^{promise} with SLOCAL complexity $O(T(n) \log n)$. As argued in Section 4.2, this implies a solution for Π with the same asymptotic SLOCAL complexity.

Let $G \in \mathcal{G}$ be the graph in which we want to solve Π^{promise}. Consider the virtual bipartite graph $\widehat{G}$ obtained by contracting each octopus gadget into a single node (see Figure 2b), that is:

- For each octopus gadget g of G, there is a white node v_g in $\widehat{G}$.
- For each inter-cluster node b of G, there is a black node u_b in $\widehat{G}$.
- For each edge connecting an inter-cluster node b of G to an octopus gadget g of G, there is an edge between v_g and u_b in $\widehat{G}$.

Note that $\widehat{G}$ may contain parallel edges. Since the diameter of a valid octopus gadget is clearly upper bounded by $O(\log n)$, we get that the distances in G are at most an $O(\log n)$ factor larger than distances in $\widehat{G}$. Thus, it is possible to simulate the execution of an SLOCAL algorithm for $\widehat{G}$ with an $O(\log n)$ multiplicative overhead on G. We use $\mathcal{A}$ to solve $\Pi^{\mathsf{linearizable}}$ on $\widehat{G}$. For each octopus gadget g, we assign the solution of the i-th port of g_v to the nodes of the i-th port gadget of g, according to the natural left-to-right order of the port gadgets of g. The output clearly satisfies the constraints of Π^{promise}, and the runtime is upper bounded by $O(T(n) \log n)$. ◀

Proof of Lemma 20. By hypothesis, there exists a non-signaling outcome O that solves Π with failure probability $p(n)$. Consider any input hypergraph F for $\Pi^{\mathsf{linearizable}}$ of size n, and let G be the bipartite incidence graph of F. We construct a graph G' as a function of G as follows.

- For each white node v of degree d of G, we put an octopus gadget g_v with d port gadgets, each of height $\Theta(\log n)$, into G'.
- For each black node u of G, we put an inter-octopus node b_u in G'.
- Let v be an arbitrary white node in G, and let $\{v, u\}$ be its i-th incident edge, according to the given ordering. We put, in G', an edge connecting b_u to the left-most leaf of the i-th port gadget of g_v, according to the natural left-to-right order of the port gadgets of g_v.

By construction, G' is a proper instance, and by Lemma 15 it can be labeled such that $G' \in \mathcal{G}$. In the following, we assume that G' is labeled in such a way. Hence, G' is an input instance for Π^{promise}. Moreover, we get that if G has n nodes, then G' has $n \leq N \leq n^3$ nodes.

We define a function f_G as follows. Let v be a white node of G, and let r be the root node of the i-th port gadget of g_v. The function f_G maps r to the i-th edge incident to v, according to the given ordering. It is straightforward to see that f_G maps solutions for Π^{promise} on G' to solutions for $\Pi^{\mathsf{linearizable}}$ on G.

Let $V_r(G')$ be the domain of f_G, and let $\Sigma_{\mathsf{promise}}$ be the set of output labels for Π^{promise} for the nodes in $V_r(G')$. Let $\{(\mathrm{out}_i, p_i)\}_{i \in I}$ be the output distribution that O defines on G' for Π. As discussed in Section 4.2, this is also a valid output distribution for Π^{promise}. Note that G is a bipartite incidence graph and $\Pi^{\mathsf{linearizable}}$ asks only to label edges of G. This means that only half-edges of F are labeled. We now define an outcome O′ on G just by describing output labelings on edges of G (which correspond to half-edges of F) – more specifically, we set $\mathrm{O}'(G) = \{(\mathrm{out}_i \circ f_G^{-1}, p_i)\}_{i \in I}$.

First, it is clear that the sum of all p_i in $\mathrm{O}'(G)$ is exactly 1. Furthermore, it is straightforward to check that O′ has failure probability at most $p(n) > 0$. If not, by construction of O′, then $\mathrm{O}(G')$ has failure probability strictly greater than $p(n)$ for Π^{promise}, which is a contradiction because O has failure probability $p(N) \leq p(n)$ by monotonicity of p.

We now claim that O′ is non-signaling beyond distance $T'(n)$. Recall that each octopus gadget in G' represents a white node v of G, and neighboring octopus gadgets represent neighboring white nodes of G. Hence, for every subset of edges A of $E(G)$, the distribution $\mathrm{O}'(G)[A]$ is defined only by $\{(\mathrm{out}_i, p_i)\}_{i \in I}[f_G^{-1}(A)]$. Let $k(n)$ be the height of a port gadget, and observe that $k(n) = \Theta(\log n)$. Suppose we modify G outside the radius-$T'(n)$ view of A and obtain a graph H with a subset of edges A_H such that $\mathcal{V}_0(A)$ is isomorphic to $\mathcal{V}_0(A_H)$ and $\mathcal{V}_{T'(n)}(A)$ is isomorphic to $\mathcal{V}_{T'(n)}(A_H)$. Notice that, as before, H also defines a proper instance $H' \in \mathcal{G}$ for Π^{promise}. However, because of the isomorphic regions between G and H, we get that $\mathcal{V}_0(f_G^{-1}(A))$ is isomorphic to $\mathcal{V}_0(f_H^{-1}(A_H))$, and $\mathcal{V}_{T'(n)\cdot k(n)}(f_G^{-1}(A))$ is isomorphic to $\mathcal{V}_{T'(n)\cdot k(n)}(f_H^{-1}(A_H))$, since each port gadget has height at least $k(n)$. We impose that $T'(n) \cdot k(n) \geq T(N)$, which is equivalent to asking that $T'(n) \geq T(N)/k(n)$. Since the distribution O is non-signaling beyond distance $T(N)$, we have that $\mathrm{O}(G')[\mathcal{V}_0(f_G^{-1}(A))]$ is the same distribution as $\mathrm{O}(H')[\mathcal{V}_0(f_H^{-1}(A))]$. Hence, $\mathrm{O}'(G)[A]$ and $\mathrm{O}'(H)[A]$ are equal, and O′ is non-signaling beyond distance $T'(n)$. Note that it is sufficient to take $T'(n) = O(T(n^3)/\log n)$, since $T(N)$ is non-decreasing in N and $n \leq N \leq n^3$. ◀

C KMW in a Nutshell

To instantiate our construction, we use a lower bound for approximating maximum matchings based on the KMW bound [16]. For completeness, we now describe the high-level idea of this bound, state it formally, and sketch the key components of its construction that are relevant for our work. See Coupette and Lenzen [11] for a detailed exposition and a simplified proof.

The KMW bound establishes that there exist graphs with n nodes and maximum degree $\Delta = 2^{\Theta(\sqrt{\log n \log\log n})}$ on which $\Omega(\sqrt{\log n / \log\log n})$ (expected) communication rounds are required to obtain polylogarithmic approximations to a minimum vertex cover, minimum dominating set, or maximum matching.

▶ **Theorem 27** ([16]). *There does not exist a randomized LOCAL algorithm providing an $O(\log \Delta)$-approximation of fractional maximum matching with locality $o(\sqrt{\log n / \log\log n})$ on graphs with degree $\Delta = 2^{\Theta(\sqrt{\log n \log\log n})}$.*

The KMW bound holds under both randomization and approximation, and it extends to symmetry-breaking tasks like finding maximal independent sets or maximal matchings via straightforward reductions.

At the core of the bound lies a class of high-girth graphs constructed from a blueprint, the *Cluster Tree*, which arranges differently-sized independent sets of nodes as a tree and prescribes that node sets adjacent in the tree are connected via biregular bipartite graphs. Both blueprints and graphs are parametrized by the number of communication rounds k, and they are designed to enable an *indistinguishability argument*: For a given k, the associated Cluster Tree graph contains two independent sets of nodes, one large and one small, such that both sets of nodes have isomorphic radius-k neighborhoods, but only the small set of nodes is needed to solve a given *covering* problem. This forces any algorithm to select a large fraction of the large node set into the solution (in expectation), yielding a poor approximation ratio.

The construction extends to *packing* problems by taking two copies of a Cluster Tree and additionally prescribing that each node in the first copy is connected to its counterpart in the second copy. Importantly, the graphs arising from Cluster Trees are bipartite by design. In bipartite graphs, the optimal fractional solution and the optimal integral solution coincide for both minimum vertex cover and maximum matching, and by Kőnig's theorem [13], the solution sets to both problems have the same cardinality. Moreover, the two-copy construction of Cluster Trees for packing problems has a natural bicoloring such that the large cluster in the first copy and the small cluster in the second copy have the same color – i.e., providing the bicoloring keeps the indistinguishability argument intact. Hence, the KMW bound is inherently a bound for (bipartite) fractional problems.

Distributed Computation with Local Advice

Alkida Balliu
Gran Sasso Science Institute, L'Aquila, Italy

Sebastian Brandt
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Fabian Kuhn
University of Freiburg, Germany

Krzysztof Nowicki
Unaffiliated

Dennis Olivetti
Gran Sasso Science Institute, L'Aquila, Italy

Eva Rotenberg
IT University of Copenhagen, Denmark

Jukka Suomela
Aalto University, Finland

Abstract

Algorithms with advice have received ample attention in the distributed and online settings, and they have recently proven useful also in dynamic settings. In this work we study *local computation with advice*: the goal is to solve a graph problem Π with a distributed algorithm in $T(\Delta)$ communication rounds, for some function T that only depends on the maximum degree Δ of the graph, and the key question is how many bits of advice per node are needed.

Some of our results regard *Locally Checkable Labeling problems* (LCLs), which is an important family of problems that includes various coloring and orientation problems on finite-degree graphs. These are constraint-satisfaction graph problems that can be defined with a finite set of valid input/output-labeled neighborhoods.

Our main results are:

1. Any *locally checkable labeling problem* can be solved with only 1 bit of advice per node in graphs with *sub-exponential growth* (the number of nodes within radius r is sub-exponential in r; for example, grids are such graphs). Moreover, we can make the set of nodes that carry advice bits arbitrarily sparse. As a corollary, any locally checkable labeling problem admits a *locally checkable proof* with 1 bit per node in graphs with sub-exponential growth.
2. The assumption of sub-exponential growth is complemented by a conditional lower bound: assuming the *Exponential-Time Hypothesis*, there are locally checkable labeling problems that cannot be solved in general with any constant number of bits per node.
3. In any graph we can find an *almost-balanced orientation* (indegrees and outdegrees differ by at most one) with 1 bit of advice per node, and again we can make the advice arbitrarily sparse. As a corollary, we can also *compress an arbitrary subset of edges* so that a node of degree d stores only $d/2 + 2$ bits, and we can *decompress* it locally, in $T(\Delta)$ rounds.
4. In any graph of maximum degree Δ, we can find a Δ-coloring (if it exists) with 1 bit of advice per node, and again, we can make the advice arbitrarily sparse.
5. In any 3-colorable graph, we can find a 3-coloring with 1 bit of advice per node. As a corollary, in bounded-degree graphs there is a locally checkable proof that certifies 3-colorability with 1 bit of advice per node, while prior work shows that this is not possible with a *proof labeling scheme* (PLS), which is a more restricted setting where the verifier can only see up to distance 1.

Our work shows that for many problems the key threshold is not whether we can achieve 1 bit of advice per node, but whether we can make the advice arbitrarily sparse. To formalize this idea, we develop a general framework of *composable* schemas that enables us to build algorithms for local computation with advice in a modular fashion: once we have (1) a schema for solving Π_1 and (2) a schema for solving Π_2 assuming an oracle for Π_1, we can also compose them and obtain (3) a schema



© Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Krzysztof Nowicki, Dennis Olivetti, Eva Rotenberg, and Jukka Suomela;
licensed under Creative Commons License CC-BY 4.0

39th International Symposium on Distributed Computing (DISC 2025).
Editor: Dariusz R. Kowalski; Article No. 12; pp. 12:1–12:19



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that solves Π_2 without the oracle. It turns out that many natural problems admit composable schemas, all of them can be solved with only 1 bit of advice, and we can make the advice arbitrarily sparse.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Distributed computing models; Mathematics of computing → Graph algorithms

Keywords and phrases Distributed graph algorithms, LOCAL model, computation with advice, locally checkable labeling problems, proof labeling schemes, locally checkable proofs, graph coloring, exponential-time hypothesis

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.12

Related Version *Full Version*: `https://arxiv.org/abs/2405.04519`

Funding This work was supported by the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI, the PNRR MIUR research project GAMING "Graph Algorithms and MinINg for Green agents" (PE0000013, CUP D13C24000430001), the Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis" and the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems".

Acknowledgements This work was initiated at Dagstuhl Seminar 23071: "From Big Data Theory to Big Data Practice". We would like to thank Sameep Dahal, Laurent Feuilloley, and Augusto Modanese for insightful discussions, and the anonymous reviewers for their helpful comments on previous versions of this work.

1 Introduction

Our work explores *what can and cannot be computed locally with the help of advice*. Our main focus is understanding advice in the context of classic local graph problems, such as vertex coloring.

While computation with different forms of advice has been explored in a wide range of distributed settings [24, 23, 14, 31, 13, 35, 46, 48, 25, 36, 51, 22, 39, 47, 26], there is hardly any prior work on solving classic local graph problems. A rare example of prior work is [20] from 2007, which studied the question of how much advice is necessary to break Linial's [44] lower bound for coloring cycles.

We initiate a systematic study of exactly how much advice is needed in the context of a wide range of graph problems. As we will see in this work, the exploration of the advice complexity of graph problems opens up connections with many other topics – it is linked with *distributed proofs* [40, 41, 42, 43, 34, 17], *distributed decompression*, the notion of *order-invariant algorithms* [50], and also with the *exponential-time hypothesis* [37] in computational complexity theory.

1.1 Local computation with advice

Let us first formalize the setting we study:

> A graph problem Π can be *solved with β bits of advice* if there exists a $T(\Delta)$-round distributed algorithm $\mathcal{A}$, such that for any graph G that admits a solution to Π, there is an assignment of β-bit labels on vertices, such that the output of $\mathcal{A}$ on the labeled graph is a solution to Π.

We note that on graphs that do not admit a solution to Π, Algorithm $\mathcal{A}$ is allowed to behave arbitrarily. For instance, when we consider the 3-coloring problem, Algorithm $\mathcal{A}$ may produce an arbitrary output on graphs that are not 3-colorable.

We will work in the usual LOCAL model of distributed computing. In an n-node graph, the nodes are labeled with unique identifiers from $\{1, 2, \ldots, \mathrm{poly}(n)\}$. We emphasize that the advice may depend on the assignment of identifiers, and algorithm $\mathcal{A}$ can freely make use of both the advice and the identifiers.

Now we seek to understand this question:

> What is the smallest β such that Π can be solved with β bits of advice?

Note that if Π is, for example, the 3-coloring problem, it is trivial to solve with $\beta = 2$ bits of advice per node, as we can directly encode the solution. The key question is how much better we can do.

Our work shows that for many problems the key threshold is not whether we can achieve 1 bit of advice per node, but whether we can make the advice *arbitrarily sparse*, that is, make the ratio between 1s and 0s assigned to the nodes of the graph to be an arbitrarily small constant. This is particularly useful, as it enables us to *compose* multiple sparse advice schemes, so that it suffices to use just one bit per node in total (we will elaborate on this in Section 1.7). Hence, a large part of this paper addresses the following question:

> Which problems admit arbitrarily sparse advice?

The notion of the sparsity of the advice is discussed in more detail later in the paper. In this context, our paper shows that some problems can be solved with arbitrarily sparse advice. On the other hand, we also show that assuming the Exponential-Time Hypothesis, for any constant c, there exist problems that cannot be solved with c bits of advice. Finally, there are some problems like 3-coloring that can be solved with 1 bit of advice, but where it is not clear whether it can be solved with arbitrarily sparse advice. We discuss all of those points in more detail in the remaining part of this section; we refer to the full version of this work for the omitted proofs.

1.2 Contribution 1: LCLs in bounded-growth graphs

Locally checkable labeling problems (LCL), first introduced by Naor and Stockmeyer [50] in the 1990s, are one of the most extensively studied families of problems in the theory of distributed graph algorithms. These are graph problems that can be specified by giving a *finite* set of valid local neighborhoods. Many key problems such as vertex coloring, edge coloring, maximal independent set, maximal matching, sinkless orientation, and many other splitting and orientation problems are examples of LCLs, at least when restricted to bounded-degree graphs. Thanks to the extensive research effort since 2016, we now understand very well the landscape of *all* LCL problems and their computational complexities across different models of distributed computing [5, 3, 11, 28, 2, 19, 54, 9, 10, 30, 4].

We design a schema that allows us to solve *any* LCL problem with just one bit of advice in graphs with a sub-exponential growth (the number of nodes in a radius-r neighborhood is sub-exponential in r):

> Any LCL problem can be solved with 1 bit of advice per node in sub-exponential growth graphs.

Furthermore, we show that the encoding can be made arbitrarily sparse. Note that e.g. grids have polynomial growth while e.g. regular trees have exponential growth, so the result is applicable in grids but not in trees.

1.2.1 Application: locally checkable proofs in bounded-growth graphs

One prominent application of this result is its connection with *distributed proofs* [40, 41, 42, 43, 17], and in particular with *locally checkable proofs* [34]. Consider any LCL Π. Assume that our task is to prepare a distributed proof that shows that in a graph G there exists a feasible solution of Π (for example, if Π is the task of 10-coloring, then the task is to certify that the chromatic number of G is at most 10). Now if G has sub-exponential growth, we can use our result to prepare a 1-bit advice that enables the algorithm to find a solution of Π. Our advice is the proof: to verify it, we simply try to recover a solution with the help of the advice, and then check that the output is feasible in all local neighborhoods (recall that Π is locally checkable). We obtain the following corollary:

> Any LCL problem admits a locally checkable proof with 1 bit per node in graphs with sub-exponential growth.

Note that this is not a proof labeling scheme as defined in [40, 41, 42, 43], as the verifier running at node u may need to see more than just the identifier and the proof label of u and the proof labels of u's immediate neighbors. However, in bounded-degree graphs it is a locally checkable proof (LCP) as defined in [34]; for a fixed Δ the verification radius is a constant $T(\Delta)$.

So to summarize, if we can solve some LCL problem Π with b bits of advice per node, then we also have an LCP for the graph property "G admits a feasible solution to Π" with b-bit proofs per node. The converse is not true: Consider the LCL problem Π that encodes the task "orient edges so that each node has indegree equal to outdegree". Now to prove "G admits a feasible solution to Π" one can use a 0-bit LCP, where even-degree nodes accept and odd-degree nodes reject. However, this LCP does not help us at all if we would like to solve Π with the help of advice. In this sense distributed computation with advice is a harder problem than local proofs.

It is also good to note that one can have LCPs for graph properties that are not of the form "G admits a feasible solution to some LCL Π." For example, planarity is such a property. Such LCPs are (to our knowledge) not directly connected with computation with local advice.

1.3 Contribution 2: LCLs in general graphs

At this point a natural question is whether the assumption about bounded growth is necessary. Could we solve all LCL problems in all graphs with 1 bit of advice? In Section 4 we show that the answer is likely to be no:

Fix any β. If all LCL problems can be solved locally with at most β bits of advice, then the Exponential-Time Hypothesis (ETH) is false.

The intuition here is that if some LCL problem Π can be solved with, say, 1 bit of advice per node with some local algorithm $\mathcal{A}$, then we could solve it with a centralized sequential algorithm as follows: check all 2^n possible assignments of advice, apply $\mathcal{A}$ to decode the advice, and see if the solution is feasible. The total running time (from the centralized sequential perspective) would be $2^n \cdot n \cdot s(n)$, where $s(n)$ is the time we need to simulate $\mathcal{A}$ at one node. Then we need to show that assuming the Exponential-Time Hypothesis, this is too fast for some LCL problem Π. However, the key obstacle is that it may be computationally expensive to simulate $\mathcal{A}$, as it might perform arbitrarily complicated calculations that depend on the numerical values of the unique identifiers, and we cannot directly bound $s(n)$.

Hence, we need to show that $\mathcal{A}$ can be made cheap to simulate. The key ingredient is the following technical result, which we prove using a Ramsey-type argument that is inspired by the proof of Naor and Stockmeyer [50]:

Assume that problem Π can be solved with β bits of advice per node, using some algorithm $\mathcal{A}$. Then the same problem can be also solved with β bits of advice using an *order-invariant* algorithm $\mathcal{A}'$, whose output does not depend on the numerical values of the identifiers but only on their relative order.

The key point here is that (for bounded-degree graphs) $\mathcal{A}'$ can be represented as a finite lookup table; hence the simulation of $\mathcal{A}'$ is cheap, and we can finally make a formal connection to the Exponential-Time Hypothesis. We refer to Section 4 for more details.

1.4 Contribution 3: balanced orientations

Now we move on to a specific graph problem: we study the task of finding balanced and almost-balanced orientations. The goal is to orient the edges so that for each node indegree and outdegree differ by at most 1. This is a hard problem to solve in a distributed setting, while slightly more relaxed versions of the problem admit efficient (but not constant-time) algorithms [29].

Here it is good to note that if we could place our advice on *edges*, then trivially one bit of advice per edge would suffice (simply use the single bit to encode whether the edge is oriented from lower to higher identifier). However, we are here placing advice on *nodes*, and encoding the orientation of each incident edge would require a number of bits proportional to the maximum degree. Surprisingly, we can do it, in any graph:

We can find almost-balanced orientations with 1 bit of advice per node.

Again, we can make the advice arbitrarily sparse.

1.4.1 Application: distributed decompression

Equipped with the advice schema for solving almost-balanced orientations, we can now make a formal connection to what we call *distributed decompression*. Here the task is to encode some graph labeling so that it can be decompressed locally (in $T(\Delta)$ rounds).

Local decompression is closely linked with local computation with advice. If we can compress some solution to Π with only β bits per node, and decompress it locally, then we can also solve Π with β bits of advice per node. Furthermore, if Π is a problem such that for any graph there is only one feasible solution, then the two notions coincide.

We will now show yet another connection between local decompression and local computation with advice. Consider the task of compressing an *arbitrary subset of edges* $X \subseteq E$. In a trivial encoding, we label each node v of degree d with a d-bit string that indicates which of the incident edges are present in X. On the other hand, we need a total of $|E|$ bits in order to distinguish all subsets of the edge-set E. In particular, for d-regular graphs, this means we need at least $d/2$ bits per node to recover an arbitrary subset of edges.

It turns out that once we can solve almost-balanced orientations, we can also compress a subset of edges efficiently. We simply use 1 bit of advice per node to encode an almost-balanced orientation. Now a node of degree d has outdegree $\delta \leq \lceil d/2 \rceil$, and it can simply store a δ-bit vector that indicates which of its outgoing edges are in X. Overall, we will need $\lceil d/2 \rceil + 1$, i.e. $\leq d/2 + 2$, bits per node:

> We can encode an arbitrary set of edges $X \subseteq E$ so that a node of degree d only needs to store $\lceil d/2 \rceil + 1$ bits, and we can decompress X locally, in $T(\Delta)$ rounds.

1.5 Contribution 4: vertex Δ-coloring

Next we study the problem of Δ-coloring graphs of maximum degree Δ:

> In any graph of maximum degree Δ, we can find a Δ-coloring (if it exists) with 1 bit of advice per node.

Again, we can make the advice arbitrarily sparse.

Our schema for encoding Δ-colorings consists of three steps. First, we compute a vertex coloring with $O(\Delta^2)$ colors, with the help of advice. Then we reduce the number of colors down to $\Delta + 1$, using the algorithm by [21, 6, 45]. Finally, we follow the key idea of the algorithm by Panconesi and Srinivasan [52] to turn $(\Delta + 1)$-coloring into a Δ-coloring, and again we will need some advice to make this part efficient.

1.6 Contribution 5: vertex 3-coloring

So far we have seen primarily results of two flavors: many problems can be solved with 1 bit of advice so that we can make the advice arbitrarily sparse, while there are also some problems that require arbitrarily many bits of advice.

We now turn our attention to a problem that seems to lie right at the boundary of what can be done with only 1 bit per node: vertex 3-coloring in any 3-colorable graph. Note that this is a problem that is hard to solve without advice not only in the distributed setting (it is a global problem) but also in the centralized setting (it is an NP-hard problem).

In the centralized setting, 1 bit of advice per node makes the problem easy. To see this, we can simply use the bit to indicate which nodes are of color 3. Then the rest of the graph has to be bipartite, and we can simply find a proper 2-coloring in polynomial time.

In the distributed setting, the trivial solution does not work: 2-coloring in bipartite graphs is still a global problem. Nevertheless, we show that 3-coloring is still doable with 1 bit of advice:

In any 3-colorable graph, we can find a 3-coloring with 1 bit of advice per node.

Our encoding essentially uses one bit to encode one of the color classes, but we adjust the encoding slightly so that throughout the graph there are *local hints* that help us to also choose the right parity for the region that we need to 2-color.

Here, our encoding genuinely needs one bit per node (it just barely suffices); we cannot make our advice arbitrarily sparse.

1.6.1 Application: locally checkable proofs for 3-coloring

Following the same idea as in Section 1.2.1, we can now also certify that a graph is 3-colorable with a proof that only takes 1 bit per node. Furthermore, the verifier only needs to see up to distance $T(\Delta)$, and hence if $\Delta = O(1)$, this is a locally checkable proof in the sense of [34]:

There is a locally checkable proof that certifies 3-colorability with 1 bit per node in graphs of maximum degree $\Delta = O(1)$.

Now it is interesting to compare this with *proof labeling schemes* (PLS). In essence, a PLS is an LCP in which the verifier only sees the identifier of the present node, and the proof labels within radius 1. The above result provides a 1-bit LCP but not a 1-bit PLS.

This connects directly with the work done by Ardévol Martínez et al. [1] and Bousquet et al. [7], which study the same question: how many bits per node are needed to certify k-colorability. Here Bousquet et al. [7] shows that $\Theta(\log k)$ bits per node are necessary for a PLS that certifies k-colorability, and [1] shows that in particular 3-colorability cannot be certified with 1 bit per node with any PLS (while 2 bits per node is trivial). Our work complements the latter result, and provides a separation between PLSs and LCPs in this setting: now we know that while 1 bit per node does not suffice to certify 3-colorability with any PLS, it is sufficient for an LCP (with the caveat that we need to be in a bounded-degree graph).

1.7 Key technique: composability framework

We already discussed some of the proof ingredients above. However, there is one additional technique that we use in many of our algorithms: the framework of *composable schemas*.

It turns out that for many problems, it is easier to work with advice schemas in which only a few nodes carry advice bits, but they may carry many bits of advice. In Definition 2 we give the formal definition of such a schema, and in Definition 4 we give the formal definition of *composable* schemas, which satisfy the additional property that the ratio between the total number of bits held by the nodes, and the total number of nodes, can be made arbitrarily small in every large-enough neighborhood.

While the definition is a bit technical, it has two key properties, which we discuss in more detail in the full version of this work:

1. As the name suggests, composable schemas can be easily composed, in the following sense: once we have (1) a composable schema for solving Π_1 and (2) a composable schema for solving Π_2 assuming an oracle for Π_1, we can also compose them and obtain (3) a composable schema that solves Π_2 without the oracle. This way we can solve problems in a modular fashion, in essence using schemas as "subroutines."

2. A composable schema can be then encoded with only 1 bit of advice per node, and we can make the advice arbitrarily sparse.

For example, our algorithms for finding almost-balanced orientations and Δ-coloring with advice are based on the framework of composable schemas.

1.8 Open questions

Our work suggests a number of open questions:

1. Our negative result in Section 4 assumes the Exponential-Time Hypothesis. Is it possible to prove an unconditional lower bound without such assumptions? Can we exhibit a concrete LCL problem Π that is unconditionally hard?
2. We conjecture that the advice for vertex 3-coloring in 3-colorable graphs cannot be made arbitrarily sparse. Is this true?
3. We also conjecture that for a sufficiently large d, vertex d-coloring in d-colorable graphs cannot be encoded with one bit per node. Is this true? This is closely linked with the discussion on the trade-off conjecture in e.g. [1, 7].
4. Our schema for distributed decompression is asymptotically optimal, but there is room for improving additive constants. Here is a concrete open question: Let $G = (V, E)$ be a 3-regular graph, and let $X \subseteq E$ be an arbitrary set of edges. Is it possible to encode X using only 2 bits per node so that it can be decompressed locally? (Note that 1 bit per node is trivially impossible, while 3 bits per node is trivial. If we delete one edge from each connected component, an encoding with 2 bits per node follows from 2-degeneracy.)

2 Additional related work

Computing with advice is not really a well-defined area of research, as one can consider many kinds of advice and various models of computations. In this paper, we focus on existential advice and the distributed LOCAL model: the advice is provided by an all-knowing oracle, and it is designed to enable algorithms with low locality (i.e., low time complexity). More broadly, one can consider a variant in which the restriction on the size of advice is more relaxed, but it needs to be computable by an oracle with some limitations (realizable in some model of computation), which then can be used as a building block of algorithms.

Advice and locality in distributed computing. Feuilloley et al. [18] consider the framework of Proof Labeling Schemes (PLS). In this framework, there is a *prover* and a *verifier*: the prover is a centralized entity that assigns labels to the nodes, while the verifier is a distributed algorithm that in T rounds is able to verify the validity of the collection of labels. If the predicate we want to investigate holds true, then there must exist an assignment of labels (or certificates) such that each node, in T rounds, "accepts" the given labeling; while if the predicate we want to investigate does not hold, then for any given labeling assignment there must exist a node that rejects it. The authors study the tradeoff between the size of the labels given at each node and the round-complexity of the verifier. In their work, they, too, discover how this tradeoff is remarkably different depending on the growth rate of the graph.

Fraigniaud et al. [20] investigate the problem of distributedly 3-coloring a cycle. Without advice, it is known that this problem requires $\Omega(\log^* n)$ rounds [44], and the paper seeks to understand whether advice can help to break this barrier. In the context studied in [20], each node receives some advice as input, and the total advice is measured as the sum of the lengths of the bit-strings given to all nodes. The authors show that, for any constant k, $O(n/\log^{(k)} n)$ bits of total advice are not sufficient to beat the $\Omega(\log^* n)$ lower bound, where $\log^{(k)} n$ denotes k recursive iterations of $\log n$.

Algorithm design. Parnas and Ron [53] show that one can use distributed algorithms to derive (by simulation) Local Computation Algorithms. Quite naturally, one can use the same approach to derive parallel graph algorithms or graph algorithms for dynamic data sets (or graph algorithms for any model of computation that can leverage the fact that the output is defined by some small neighborhood of a vertex or edge). However, as is, the reduction from [53] did not immediately imply many new state-of-the-art algorithms, as the size of the neighborhood grows exponentially with locality, and, for many problems, the locality of the fastest-known algorithms is fairly large. One of the ways to address this issue is to rely on analysis of algorithms with advice.

Christiansen et al. [12] use distributed algorithms with advice to design faster dynamic algorithms for vertex coloring, by combining dynamic graph orientations with ideas for simulating distributed algorithms on the resulting directed graph.

While this approach in principle can be used in the design of parallel algorithms, so far it is used implicitly or in a somewhat trivial way (e.g. Ghaffari et al. [27] mention that 4-coloring of a graph can be trivially used to compute a maximal independent set).

Advice in other models of computation with partial knowledge. More generally, one can consider the impact of advice that captures some global knowledge about the whole input (or distribution of inputs) on the performance of algorithms. Similarly to the case of distributed computing, one can consider existential advice and advice that can be realized in a somewhat practical model of computation.

Emek et al. [15] introduced the notion of advice in the context of online algorithms. The authors studied the impact of advice in the competitive ratio of some classical online problems, namely *metrical task systems* and *k-server*, and they show tradeoffs between the number of bits of advice and the achievable competitive ratio. We refer to the survey by Boyar et al. [8] for more work on online algorithms with advice.

Mitzenmacher and Vassilvitskii [49] show that algorithms using advice realized by machine learning models can be used to provide algorithms that on average perform better than their traditional counterparts while keeping the same worst-case bounds. However, this particular advice model allows querying an oracle at each step. As such, in the context of online algorithms, it is in some sense both weaker than the existential advice for online algorithms (as it is produced by some machine learning model) and stronger as it gives a lot of bits of advice, and has access to the prefix of the input.

3 Preliminaries

Let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. We denote with n the number of nodes of the graph, and with Δ its maximum degree. We may use the notation $V(G)$ to denote the set of nodes of G, and $E(G)$ to denote the set of edges of G. With $\mathrm{dist}_G(u, v)$ we denote the distance between $u \in V$ and $v \in V$ in G, that is, the length of the shortest path between u and v in G, where the length of a path is the number of edges of the path. For an integer k, the power graph G^k of a graph G is the graph that contains the same nodes as G, and there is an edge $\{u, v\}$ in G^k if and only if $1 \leq \mathrm{dist}_G(u, v) \leq k$. Two nodes u and v are neighbors in G if there is an edge $\{u, v\} \in E(G)$.

A maximal independent set (MIS) of G is a subset S of nodes of G satisfying that no nodes of S are neighbors in G, and that all nodes in $V \setminus S$ have at least one neighbor that is in S. An (α, β)-ruling set is a subset S of nodes of G satisfying that each pair of nodes

from S have distance at least α, and that all nodes in $V \setminus S$ have at least one node in S at distance at most β. Observe that Maximal Independent Set and $(2,1)$-ruling set is the same problem. By β-ruling set we denote the $(2,\beta)$-ruling set problem.

One important tool that we will use is the Lovász Local Lemma, which states the following.

▶ **Lemma 1** (Lovász Local Lemma [55, 56, 16]). *Let $\{E_1, \ldots, E_k\}$ be a set of events satisfying the following conditions: each event E_i depends on at most d other events; each event E_i happens with probability at most p. Then, if $epd \leq 1$, there is non-zero probability that none of the events occur.*

3.1 LOCAL model

In the LOCAL model of distributed computation, each node of an n-node graph is equipped with a unique ID, typically in $\{1, \ldots, n^c\}$, for some constant c. Initially, each node in the graph knows its own ID, its degree (i.e., the number of neighboring nodes), the maximum degree Δ in the graph, and the total number of nodes. Then, the computation proceeds in synchronous rounds where at each round each node exchanges messages with its neighbors and performs some local computation. The size of the messages can be arbitrarily large and the local computation can be arbitrarily heavy. The runtime of an algorithm is defined as the number of rounds it requires such that all nodes terminate and produce an output.

3.2 Locally checkable labelings

We formally define an LCL problem Π as a tuple $(\Sigma_{\text{in}}, \Sigma_{\text{out}}, C, r)$ where each element of the tuple is defined as follows. The parameters Σ_{in} and Σ_{out} are finite sets of input and output labels, respectively. The parameter r is a positive integer called *checkability radius* of Π, i.e., it determines how far in the graph each node needs to check in order to verify the validity of a given solution. The parameter C determines the constraints of Π, that is, C is a finite set of labeled graphs H containing a vertex of eccentricity at most r, where each edge-endpoint pair $(uv, v) \in E_H \times V_H$ has a label $\ell_{\text{in}} \in \Sigma_{\text{in}}$ and a label $\ell_{\text{out}} \in \Sigma_{\text{out}}$.

Let $\Pi = (\Sigma_{\text{in}}, \Sigma_{\text{out}}, C, r)$ be an LCL problem and let $G = (V, E)$ be a graph where each edge-endpoint pair (uv, v) is labeled with a label from Σ_{in}. The task of solving Π on G requires labeling each edge-endpoint pair $(uv, v) \in E \times V$ with a label in Σ_{out} such that, for each node $v \in V$ it holds that the graph induced by the nodes at distance at most r from v and edges that have at least one endpoint at distance at most $r-1$ from v is isomorphic to some (labeled) graph in C.

3.3 Advice schema

We now formally define the notion of advice schema.

▶ **Definition 2** (Advice Schema). *A $(\mathcal{G}, \Pi, \beta, T)$-advice schema is a function f that receives as input a (possibly input-labeled) graph $G = (V, E, I) \in \mathcal{G}$ (where I is an input for the nodes and/or the edges of the graph), and outputs a function $\ell_G := f(G)$ that satisfies the following.*

- *The function ℓ_G maps each node $v \in V$ into a bit-string of length at most β, where β is a function of Δ.*
- *There exists a LOCAL algorithm $\mathcal{A}$ that, for each $G \in \mathcal{G}$, if we label each node $v \in V$ with the bit-string $\ell_G(v)$, then $\mathcal{A}$ runs in at most T rounds and outputs a valid solution for Π, where T is a function of Δ.*

Moreover, we distinguish three possible types of advice schemas:

1. If all nodes of the graph receive bit-strings of the same length, then the schema is called *uniform fixed-length*.
2. If a subset of nodes receives bit-strings of the same length, and all the others receive bit-strings of length 0, then the schema is called *subset fixed-length*.
3. If a subset of nodes receives a bit-string of possibly different positive lengths, and all the others receive bit-strings of length 0, then the schema is called *variable-length*. Observe that an advice schema, as defined in Definition 2 and without further assumptions, is variable-length.

In the second and third cases, the nodes having non-zero bit-strings assigned are called *bit-holding* nodes. Moreover, observe that Type 1 is a special case of Type 2, which is a special case of Type 3.

For uniform fixed-length advice schemas where all nodes receive one bit, we define the notion of sparsity, which captures the fact that the ratio between nodes receiving a 1 and nodes receiving a 0 can be made arbitrarily small.

▶ **Definition 3** (Sparse schema). *A uniform fixed-length $(\mathcal{G}, \Pi, \beta, T)$-advice schema is called* ε-sparse *if the following holds:*

- *$\beta = 1$, that is, each node receives exactly one bit.*
- *For any graph $G \in \mathcal{G}$, let n_0 be the number of nodes to which the schema (i.e., the function f of Definition 2) assigns a 0, and let n_1 be the number of nodes to which f assigns a 1. Then, $\frac{n_1}{n_0+n_1} \leq \varepsilon$.*

With abuse of notation, we call a uniform fixed-length $(\mathcal{G}, \Pi, \beta, T)$-advice schema (where T is a function that receives as input ε and returns a function of Δ) sparse *if, for any constant $\varepsilon > 0$, there exists an ε-sparse $(\mathcal{G}, \Pi, \beta, T(\varepsilon))$-advice schema.*

3.4 Composability

The main idea that we will use to devise our advice schemas is the following. Given some problem Π, it may be cumbersome to directly define an advice schema for it. Instead, it may be easier to do this operation gradually. In more detail, many problems can be solved as follows: first, find a solution for some subproblem; then, use the solution for the subproblem in order to solve the problem more easily. As a running example, consider the following problem Π.

- The input is a bipartite graph where all nodes have even degree.
- It is required to output a coloring of the edges, say red and blue, such that each node has the same number of red and blue incident edges.

Consider the following three problems.

- Let Π_{v} be the problem of computing a 2-coloring of the nodes, say, black and white.
- Let Π_{e} be the problem of outputting a 2-coloring of the edges, such that each node has the same number of red and blue incident edges, assuming that we are given as input a 2-coloring of the nodes and a balanced orientation of the edges, and assuming that all nodes have even degree.
- Let Π_{o} be the problem of orienting all edges such that each node has the same number of incoming and outgoing edges, assuming that all nodes have even degree.

Consider the following algorithm. First, solve Π_{v} and Π_{o}. Then, solve Π_{e}, by coloring *red* the edges oriented from black to white, and by coloring *blue* the edges oriented from white to black. Observe that this algorithm solves Π.

In other words, we decomposed Π into three different subproblems: two of them are "hard" problems (Π_{v} and Π_{o}), for which some advice is needed if we want to solve them fast, and one (Π_{e}) is trivial once we are given as input the solution for the other two. The idea now is to devise two different advice schemas for Π_{v} and Π_{o} separately.

- For Π_{v} there is a trivial advice schema: use 1 bit to encode the 2-coloring of the nodes.
- For Π_{o} the advice schema is non-trivial, and we will show how it can be done by using 1 bit per node.

While both schemas use only 1 bit per node, we cannot directly combine these two schemas into a single schema that uses 1 bit per node. For this reason, we will devise our schemas in a more high-level way. In particular, for most of the results of this paper, we will not directly provide a uniform fixed-length advice schema that uses one bit per node, but we will start by devising variable-length schemas. Variable-length schemas provide a simpler way to encode information: we can choose a subset of nodes and assign bit-strings to them, without worrying about how these strings will be later encoded by giving one bit to each node. Moreover, the variable-length schemas that we provide satisfy a property called *composability*, which we will formally define later. Intuitively, this property requires the ability to make the set of bit-holding nodes of a variable-length schema to be arbitrarily sparse. For example, for Π_{v}, we do not really need to provide the color of all nodes: we assign 1 bit to a sparse set of nodes (encoding their color), and to all other nodes we do not assign any bit. The nodes that have no bit assigned can still recover a 2-coloring by simple propagation. For Π_{o}, things are more complicated, but a composable variable-length schema can be obtained as well.

Once we have a composable schema for all subproblems that are used to solve our problem of interest, we apply, as a black box, a lemma to obtain a single variable-length schema for our problem. Then, again as a black box, we convert such a schema into a uniform fixed-length schema that uses a single bit per node.

Summarizing, many of the results that we provide about schemas that use a single bit per node are based on the following idea:

- We first decompose a problem of interest into many subproblems;
- We show that, for each of the subproblems, it is possible to devise a variable-length schema that satisfies some desirable properties;
- We prove that such properties imply that we can compose many variable-length schemas into a single one that satisfies the same properties;
- We prove that the resulting variable-length schema implies a uniform fixed-length schema that uses a single bit per node.

Note that the last two steps are done in a problem-independent way. The conditions that allow to combine multiple variable-length schemas are expressed in Definition 4, that, on a high level, states the following. We want our variable-length schema to be tunable as a function of three parameters α, γ, and c. The schema needs to satisfy that in each α-radius neighborhood there are at most γ bit-holding nodes, and that the number of bits held by these nodes is upper bounded by $c\alpha/\gamma^3$. Hence, a composable schema is a collection of schemas, one for each choice of parameters α, γ, and c.

▶ **Definition 4** (Composability). *A $(\mathcal{G}, \Pi, \gamma_0, A, T)$-composable advice schema is a collection $\mathcal{S}$ of advice schemas satisfying the following. For any constant $c > 0$, any $\gamma \geq \gamma_0$, and any $\alpha \geq A(c, \gamma)$, there exists $\beta \leq c\alpha/\gamma^3$ such that:*

- *The collection $\mathcal{S}$ contains a variable-length $(\mathcal{G}, \Pi, \beta, T(\alpha, \Delta))$-advice schema S.*
- *For each $G \in \mathcal{G}$, the assignment given by S to the nodes of G satisfies that, in each α-radius neighborhood of G, there are at most γ_0 bit-holding nodes.*

4 Structural results and lower bounds

Our main goal here is to show that if we were able to solve all LCLs with some constant number of bits of advice per node, it would violate the *Exponential-Time Hypothesis* (ETH) [37]. However, to do that we need to first show a structural result, which is also of independent interest: algorithms that make use of advice can be made *order-invariant*, i.e., they do not need the numerical values of the identifiers.

4.1 Order-invariance

In general, in an advice schema our advice bits may depend on the numerical values of the identifiers, and the algorithm $\mathcal{A}$ that solves the problem may make use of the advice bits and the numerical values of the identifiers. However, here we will show that we can essentially for free eliminate the dependency on the numerical values of the identifiers.

A distributed algorithm is called *order-invariant* [50] if the output remains the same if we change the numerical values of the identifiers but preserve their relative order. Put otherwise, an order-invariant algorithm may make use of the graph structure, local inputs (which in our case includes the advice), and the relative order of the identifiers, but not the numerical values of the identifiers.

Naor and Stockmeyer [50] showed that constant-time distributed algorithms that solve LCL problems can be made order-invariant, and since then order-invariant algorithms have played a key role in many lower-bound results related to constant-time distributed algorithms, see e.g. [32, 33].

We say that graph problem Π is *component-wise-defined* if the following holds: a valid solution for graph G is also a valid solution for a connected component that is isomorphic to G. Put otherwise, we can always solve Π by splitting the graph into connected components and solving Π separately in each component. In essence all graph problems of interest (especially in the distributed setting) are component-wise defined, but one can come up with problems that do not satisfy this property (a trivial example being the task of outputting the number of nodes in the graph).

We say that graph problem Π is a *finite-input* problem if the nodes and edges are either unlabeled, or they are labeled with labels from some finite set Σ_{in}.

Note that Theorem 5 below is applicable to LCL problems (they are component-wise defined, and in any LCL we have a bounded Δ and bounded alphabets $\Sigma_{\text{in}}, \Sigma_{\text{out}}$), but also to many other problems that are not locally checkable. We emphasize that we will assume some bound on the maximum degree Δ, but if we have an encoding schema that works for any Δ, we can apply the following result separately for each Δ.

▶ **Theorem 5.** *Fix a maximum degree Δ and the number of advice bits β. Assume that Π is a finite-input component-wise defined graph problem, in which the task is to label nodes with labels from some finite set Σ_{out}. Assume that we can solve some graph problem Π with some distributed algorithm $\mathcal{A}$ in T rounds using β bits of advice. Then we can also solve Π with an order-invariant algorithm $\mathcal{A}'$ in T rounds using β bits of advice.*

Proof. Let E be the *encoder* that, given a graph G (with some unique identifiers), produces β-bit advice strings that $\mathcal{A}$ can then use to solve Π.

In the first steps, we follow the basic idea of Naor and Stockmeyer [50] to manipulate $\mathcal{A}$. Algorithm $\mathcal{A}$ is a mapping from labeled radius-T neighborhoods to local outputs; we write here $N^T(v)$ for the radius-T neighborhood of node v, and we let $s = \Delta^{T+1}$ be an upper bound on the number of nodes in $N^T(v)$.

In general, we can encode all information in $N^T(v)$ as follows:

1. The *set* $X \subseteq \{1, 2, \dots\}$ *of unique identifiers* present in the neighborhood, with $|X| \leq s$.
2. The *structure* S of the neighborhood, which includes the graph topology, the relative order of the identifiers, local inputs, and the advice bits.

So we can reinterpret $\mathcal{A}$ as a function that maps a pair (X, S) to the local output $\mathcal{A}(X, S) \in \Sigma_{\text{out}}$. But we can equally well interpret $\mathcal{A}$ as a function A that maps X to a *type* $A(X) = F$, where F is a function that maps S to the local output $F(S) \in \Sigma_{\text{out}}$; we simply let $A(X)(S) = \mathcal{A}(X, S)$. The second interpretation turns out to be convenient.

Notice that we can always pad X with additional identifiers (that use values larger than any value in X), so that we will have $|X| = s$.

Then we make the key observation: *there are only finitely many different structures* S. This follows from the assumptions that Δ is a constant, β is a constant, Π is a finite-input problem, and T is a constant. Furthermore, Σ_{out} is finite, so F is a mapping from a finite set to a finite set, and follows that *there are only finitely many different types* F. Let m be the number of possible types, and identify the types with $1, 2, \dots, m$.

With this interpretation, A defines a *coloring* (in the Ramsey-theoretic sense) of all s-subsets of natural numbers with m colors, that is, it assigns to each subset $X \subseteq \{1, 2, \dots\}$ with $|X| = s$ some *color* $A(X) \in \{1, 2, \dots, m\}$. By applying the infinite multigraph version of Ramsey's theorem, it follows that there exists an infinite set of natural numbers I and a single *canonical type* F^* such that for any $X \subseteq I$ we have $A(X) = F^*$. Set I is called a *monochromatic set*.

Now let us stop for a moment and digest what we have learned so far: if our unique identifiers came from the set I, then we will have $\mathcal{A}(X, S) = A(X)(S) = F^*(S)$. That is, $\mathcal{A}$ will then ignore the numerical values of the identifiers and only pay attention to the structure S. However, this is a big if; in general our identifiers can be arbitrary, and even adversarial.

Let us now continue; we will now modify the encoder E. We construct a new encoder E' that works as follows. Assume we are given a graph G, together with some assignment of unique identifiers. The encoder E' first constructs graph G_1 by renumbering the identifiers (but preserving their order) so that all identifiers in G_1 come from the monochromatic set I.

Now we would like to apply encoder E to G_1, but we cannot. Our encoder may assume that the identifiers in an n-node graph come from the set $\{1, 2, \dots, \text{poly}(n)\}$, while now we have made our identifiers astronomically large. But to fix this we exploit the fact that Π is component-wise solvable. We simply construct a new graph H that consists of sufficiently many copies of G, such that the first copy is G_1, with unique identifiers coming from I, while in the other copies we assign identifiers from $\{1, 2, \dots\} \setminus I$. This way we can arrange things so that H has N nodes, for some (very large) number N, and the identifiers are assigned from $\{1, 2, \dots, N\}$.

Now H is a valid instance, and we can feed it to the encoder E, which will label it with β-bit labels so that if we apply $\mathcal{A}$ to H and these labels, we will correctly solve Π in each component of H. In particular, we will correctly solve Π in G_1. Moreover, in component G_1, algorithm $\mathcal{A}$ will apply order-invariant algorithm F^* in all neighborhoods, ignoring the numerical values of the identifiers.

However, we needed an encoding for our *original* graph G, with the *original* set of identifiers. To do that, we proceed as follows. We make the above thought experiment, to construct the advice for H. Then we simply copy the advice bits from component G_1 to the original graph G.

If we now applied $\mathcal{A}$ to solve Π in G, it does not necessarily work. However, if we apply $\mathcal{A}'(X, S) = F^*(S)$ to solve Π in G, it will behave in exactly the same way as applying $\mathcal{A}$ in G_1. Hence, $\mathcal{A}'$ will also solve Π correctly in G. Furthermore, $\mathcal{A}'$ is by construction order-invariant. ◀

We note that the encoder E' constructed above is not practical or efficient (even if the original encoder E is). As we will see next, merely knowing that $\mathcal{A}'$ exists can be sufficient.

4.2 Hardness assuming the Exponential-Time Hypothesis

In the full version of this work, we show that in graphs with sub-exponential growth, any LCL can be solved with one bit of advice per node. We now show that one bit does not suffice in general, assuming the Exponential-Time Hypothesis [37].

Recall that the Exponential-Time Hypothesis states that there is a positive constant $\delta > 0$ such that 3-SAT cannot be solved in time $O(2^{\delta n})$, where n is the number of variables in the 3-SAT instance.

▶ **Theorem 6.** *Fix any β. Assume that for every LCL problem Π there is some T such that, on any input, Π can be solved in T rounds with β bits of advice. Then the Exponential-Time Hypothesis is false.*

Proof. Suppose there is some β such that all LCL problems can be solved with β bits of advice. We show how to then solve 3-SAT in time $O(2^{\varepsilon n} \cdot n \cdot f(\varepsilon, \beta))$ for an arbitrarily small $\varepsilon > 0$, violating the Exponential-Time Hypothesis.

Consider some 3-SAT instance ϕ, with n variables and m clauses. By applying the sparsification lemma by [38], we can assume w.l.o.g. that $m = O(n)$.

Now turn ϕ into a bipartite graph, with nodes representing variables on one side and nodes representing clauses on the other side. Each clause is connected by edges with the three variables it contains. Each node is labeled by its type (variable or clause) and each edge vc is labeled to indicate whether the variable v is negated in the clause c. This results in a bipartite graph with $n + m = O(n)$ nodes, and each clause-node has degree at most 3. Hence, the total number of edges is at most $3m$.

Variable-nodes may have arbitrarily high degrees, but the sum of their degrees is bounded by $3m$. We replace each variable-node that has degree $d > 3$ by a cycle of d variable-nodes, with the new edges labeled by equality constraints. This results in a graph in which all nodes have degree at most 3, and the total number of nodes is bounded by $3m + m = O(n)$. Let G_0 be the resulting graph, let $n_0 = O(n)$ be the number of nodes in G_0, let $\Delta_0 = 3$ be the maximum degree, and let $\ell_0 = O(1)$ be the number of node and edge labels that we used to encode the instance.

Now it is easy to define an LCL problem Π_0 such that a solution of Π_0 in graph G_0 can be interpreted as a satisfying assignment of the variables in formula ϕ, and vice versa.

Now assume that we have defined an LCL problem Π_i and a graph G_i with n_i nodes, maximum degree Δ_i, and ℓ_i labels; the base case $i = 0$ was presented above. Define a new LCL problem Π_{i+1} and a new graph G_{i+1} as follows. We contract edges in G_i to construct G_{i+1} so that we satisfy two properties: each node in G_{i+1} represents $O(1)$ nodes of G_i, but the number of nodes is $n_{i+1} \leq n_i/2$. This can be achieved by e.g. greedily contracting edges, favoring the edges whose endpoints currently represent the smallest number of nodes of G_i.

We label the nodes of G_{i+1} so that given the input labels of the nodes, we can also recover the original graph G_i. As each node represents a bounded number of original nodes, and the original graph had maximum degree Δ_i, the new graph will have maximum degree Δ_{i+1} that only depends on i. Also, the number of labels ℓ_{i+1} will be bounded by a constant that only depends on i. To see that everything can be indeed encoded with constant-size labels, note that each new node v in G_{i+1} can have its own *local numbering* of the original nodes v' that were contracted to v, that is, each node v' can be represented as a pair (v, a), where v is a new node and a is a sequence number. This way, the new label of node v can use

constant-size triples of the form (a, b, x) to encode that graph G_i had an edge from (v, a) to (v, b) with label x, and the new edge label of edge (u, v) can also use constant-size triples of the form (a, b, x) to encode that graph G_i had an edge from (u, a) to (v, b) with label x. Finally, the new node label of v will use constant-size pairs of the form (a, x) to encode that the node label of (v, a) in G_i was x. This way we can use constant-size node and edge labels in G_{i+1} to encode the structure and node and edge labels of G_i.

Now we can define an LCL problem Π_{i+1} such that a valid solution of Π_{i+1} in G_{i+1} can be mapped to a valid solution of Π_i in G_i, and hence eventually to a satisfying assignment of ϕ, and conversely a satisfying assignment of ϕ can be turned into a valid solution of Π_{i+1}. In essence, the output label of a node in G_{i+1} captures the output labels of all $O(1)$ nodes of G_i that it represents.

Continuing this way for $\Theta(\log(\beta/\varepsilon))$ steps, we can construct a graph G_i with fewer than $\varepsilon n/\beta$ nodes. Furthermore, there is some LCL problem Π_i such that ϕ is a yes-instance if and only if there is a valid solution of Π_i in G_i.

By assumption, there exists a distributed algorithm $\mathcal{A}$ that solves Π_i with β bits of advice per node. Using Theorem 5, we can also assume that $\mathcal{A}$ is order-invariant. Hence, $\mathcal{A}$ is a finite function, and we can compute $\mathcal{A}$ in constant time (where the constant depends on i, which depends on β and ε, but is independent of n).

Now, we simply try out all possible strings of advice; there are at most $2^{\varepsilon n}$ such strings. For each advice combination, we try to apply $\mathcal{A}$ to solve Π_i in G_i; we simulate $\mathcal{A}$ at each of the $n_i = O(n)$ nodes. If and only if ϕ is satisfiable, we will find an advice string such that $\mathcal{A}$ succeeds in solving Π_i. The overall running time is $O(2^{\varepsilon n} \cdot n \cdot f(\varepsilon, \beta))$. ◀

References

1. Virginia Ardévol Martínez, Marco Caoduro, Laurent Feuilloley, Jonathan Narboni, Pegah Pournajafi, and Jean-Florent Raymond. A lower bound for constant-size local certification. *Theoretical Computer Science*, 971:114068, 2023. doi:10.1016/J.TCS.2023.114068.
2. Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. How much does randomness help with locally checkable problems? In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 299–308. ACM, 2020. doi:10.1145/3382734.3405715.
3. Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. *Distributed Computing*, 34(4):259–281, 2021. doi:10.1007/S00446-020-00375-2.
4. Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally checkable labelings with small messages. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.8.
5. Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1307–1318. ACM, 2018. doi:10.1145/3188745.3188860.
6. Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta + 1)$-coloring and applications. *Journal of the ACM*, 69(1):5:1–5:26, 2022. doi:10.1145/3486625.
7. Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun. Local certification of local properties: Tight bounds, trade-offs and new parameters. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand,*

France, volume 289 of *LIPIcs*, pages 21:1–21:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.21.

8 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys*, 50(2):19:1–19:34, 2017. doi:10.1145/3056461.

9 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 479–488. ACM, 2016. doi:10.1145/2897518.2897570.

10 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM Journal on Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

11 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM Journal on Computing*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

12 Aleksander Bjørn Grodt Christiansen, Krzysztof Nowicki, and Eva Rotenberg. Improved dynamic colouring of sparse graphs. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1201–1214. ACM, 2023. doi:10.1145/3564246.3585111.

13 Dariusz Dereniowski and Andrzej Pelc. Drawing maps with advice. *Journal of Parallel and Distributed Computing*, 72(2):132–143, 2012. doi:10.1016/J.JPDC.2011.10.004.

14 Stefan Dobrev, Rastislav Královic, and Euripides Markou. Online graph exploration with advice. In Guy Even and Magnús M. Halldórsson, editors, *Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012, Reykjavik, Iceland, June 30-July 2, 2012, Revised Selected Papers*, volume 7355 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2012. doi:10.1007/978-3-642-31104-8_23.

15 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 427–438. Springer, 2009. doi:10.1007/978-3-642-02927-1_36.

16 P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vols. I, II, III*, volume Vol. 10 of *Colloq. Math. Soc. János Bolyai*, pages 609–627. North-Holland, Amsterdam-London, 1975.

17 Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision, 2017. doi:10.48550/arXiv.1606.04434.

18 Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Computing*, 34(2):113–132, 2021. doi:10.1007/S00446-020-00386-Z.

19 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.18.

20 Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, 2009. doi:10.1007/S00446-008-0076-Y.

21 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016,*

9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pages 625–634. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.73.

22 Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Information and Computation*, 206(11):1276–1287, 2008. doi:10.1016/J.IC.2008.07.005.

23 Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *Journal of Computer and System Sciences*, 76(3-4):222–232, 2010. doi:10.1016/J.JCSS.2009.07.002.

24 Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST computation with short advice. In Phillip B. Gibbons and Christian Scheideler, editors, *SPAA 2007: Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, California, USA, June 9-11, 2007*, pages 154–160. ACM, 2007. doi:10.1145/1248377.1248402.

25 Emanuele G. Fusco and Andrzej Pelc. Trade-offs between the size of advice and broadcasting time in trees. *Algorithmica*, 60(4):719–734, 2011. doi:10.1007/S00453-009-9361-9.

26 Emanuele G. Fusco, Andrzej Pelc, and Rossella Petreschi. Topology recognition with advice. *Information and Computation*, 247:254–265, 2016. doi:10.1016/J.IC.2016.01.005.

27 Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC algorithms for mis, matching, and coloring on trees and beyond. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 34:1–34:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.34.

28 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 662–673. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00069.

29 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Yannic Maus, Jukka Suomela, and Jara Uitto. Improved distributed degree splitting and edge coloring. *Distributed Computing*, 33(3-4):293–310, 2020. doi:10.1007/S00446-018-00346-8.

30 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017. doi:10.1137/1.9781611974782.166.

31 Christian Glacet, Avery Miller, and Andrzej Pelc. Time vs. information tradeoffs for leader election in anonymous trees. *ACM Transactions on Algorithms*, 13(3):31:1–31:41, 2017. doi:10.1145/3039870.

32 Mika Göös, Juho Hirvonen, and Jukka Suomela. Lower bounds for local approximation. *Journal of the ACM*, 60(5):39:1–39:23, 2013. doi:10.1145/2528405.

33 Mika Göös, Juho Hirvonen, and Jukka Suomela. Linear-in-Δ lower bounds in the LOCAL model. *Distributed Computing*, 30(5):325–338, 2017. doi:10.1007/S00446-015-0245-8.

34 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016. doi:10.4086/TOC.2016.V012A019.

35 Barun Gorain and Andrzej Pelc. Deterministic graph exploration with advice. *ACM Transactions on Algorithms*, 15(1):8:1–8:17, 2019. doi:10.1145/3280823.

36 David Ilcinkas, Dariusz R. Kowalski, and Andrzej Pelc. Fast radio broadcasting with advice. *Theoretical Computer Science*, 411(14-15):1544–1557, 2010. doi:10.1016/J.TCS.2010.01.004.

37 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, pages 237–240. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766282.

38 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/JCSS.2001.1774.

39 Dennis Komm, Rastislav Královic, Richard Královic, and Jasmin Smula. Treasure hunt with advice. In Christian Scheideler, editor, *Structural Information and Communication*

Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings, volume 9439 of *Lecture Notes in Computer Science*, pages 328–341. Springer, 2015. doi:10.1007/978-3-319-25258-2_23.

40 Amos Korman and Shay Kutten. On distributed verification. In Soma Chaudhuri, Samir R. Das, Himadri S. Paul, and Srikanta Tirthapura, editors, *Distributed Computing and Networking, 8th International Conference, ICDCN 2006, Guwahati, India, December 27-30, 2006*, volume 4308 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2006. doi:10.1007/11947950_12.

41 Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007. doi:10.1007/S00446-007-0025-1.

42 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/S00446-010-0095-3.

43 Amos Korman, David Peleg, and Yoav Rodeh. Constructing labeling schemes through universal matrices. *Algorithmica*, 57(4):641–652, 2010. doi:10.1007/S00453-008-9226-7.

44 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

45 Yannic Maus and Tigran Tonoyan. Linial for lists. *Distributed Computing*, 35(6):533–546, 2022. doi:10.1007/S00446-022-00424-Y.

46 Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theoretical Computer Science*, 608:190–198, 2015. doi:10.1016/J.TCS.2015.09.025.

47 Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *Journal of Parallel and Distributed Computing*, 83:159–167, 2015. doi:10.1016/J.JPDC.2015.06.004.

48 Avery Miller and Andrzej Pelc. Election vs. selection: How much advice is needed to find the largest node in a graph? In Christian Scheideler and Seth Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 377–386. ACM, 2016. doi:10.1145/2935764.2935772.

49 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020. doi:10.1017/9781108637435.037.

50 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

51 Nicolas Nisse and David Soguet. Graph searching with advice. *Theoretical Computer Science*, 410(14):1307–1318, 2009. doi:10.1016/J.TCS.2008.08.020.

52 Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 581–592. ACM, 1992. doi:10.1145/129712.129769.

53 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007. doi:10.1016/J.TCS.2007.04.040.

54 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020. doi:10.1145/3357713.3384298.

55 James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985. doi:10.1007/BF02579368.

56 Joel Spencer. Asymptotic lower bounds for ramsey functions. *Discrete Mathematics*, 20:69–76, 1977. doi:10.1016/0012-365X(77)90044-9.

Towards Fully Automatic Distributed Lower Bounds

Alkida Balliu
Gran Sasso Science Institute, L'Aquila, Italy

Sebastian Brandt
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Fabian Kuhn
University of Freiburg, Germany

Dennis Olivetti
Gran Sasso Science Institute, L'Aquila, Italy

Joonatan Saarhelo
Unaffiliated, Helsinki, Finland

Abstract

In the past few years, a successful line of research has led to lower bounds for several fundamental local graph problems in the distributed setting. These results were obtained via a technique called *round elimination*. On a high level, the round elimination technique can be seen as a recursive application of a function that takes as input a problem Π and outputs a problem Π' that is one round easier than Π. Applying this function recursively to concrete problems of interest can be highly nontrivial, which is one of the reasons that has made the technique difficult to approach. The contribution of our paper is threefold.

Firstly, we develop a new and fully automatic method for finding so-called *fixed point relaxations* under round elimination. The detection of a non-0-round solvable fixed point relaxation of a problem Π immediately implies lower bounds of $\Omega(\log_\Delta n)$ and $\Omega(\log_\Delta \log n)$ rounds for deterministic and randomized algorithms for Π, respectively.

Secondly, we show that this automatic method is indeed useful, by obtaining lower bounds for defective coloring problems. More precisely, as an application of our procedure, we show that the problem of coloring the nodes of a graph with 3 colors and defect at most $(\Delta-3)/2$ requires $\Omega(\log_\Delta n)$ rounds for deterministic algorithms and $\Omega(\log_\Delta \log n)$ rounds for randomized ones. Additionally, we provide a simplified proof for an existing defective coloring lower bound. We note that lower bounds for coloring problems are notoriously challenging to obtain, both in general, and via the round elimination technique.

Both the first and (indirectly) the second contribution build on our third contribution: a new method to compute the one-round easier problem Π' in the round elimination framework. This method heavily simplifies the usage of the round elimination technique, and in fact it has been successfully exploited in a recent work in order to prove quantum advantage in the distributed setting [STOC '25].

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases round elimination, lower bounds, defective coloring

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.13

Related Version *Full Version*: https://arxiv.org/abs/2410.20224

Funding This work was partially supported by the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI, the European Union - NextGenerationEU under the Italian MUR National Innovation Ecosystem grant ECS00000041 - VITALITY – CUP: D13C21000430001, and the PNRR MIUR research project GAMING "Graph Algorithms and MinING for Green agents" (PE0000013, CUP D13C24000430001).



© Alkida Balliu, Sebastian Brandt, Fabian Kuhn, Dennis Olivetti, and Joonatan Saarhelo;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

In the standard setting of distributed graph algorithms, known as the LOCAL model [29, 35], the nodes V of a graph $G = (V, E)$ communicate over the edges E of G in synchronous rounds. Initially, the nodes do not know anything about G (except for their own unique identifier and possibly some global parameters such as the number of nodes n or the maximum degree Δ) and at the end, each node must output its local part of the solution for the graph problem that needs to be solved. For example, if we intend to compute a vertex coloring of G, at the end, every node must output its own color in the final coloring. The time complexity of such a distributed algorithm is then measured as the number of rounds needed from the start until all nodes have terminated.

The study of the complexity of solving graph problems in the LOCAL model and in related distributed models has been a highly active area of research with a variety of substantial results over the last years. Apart from very significant and insightful new algorithmic results for distributed graph problems (e.g., [20, 19, 26, 36, 21, 25, 24]), the last ten years in particular also brought astonishing progress on proving lower bounds for distributed graph problems in the LOCAL model (e.g., [17, 19, 3, 5]). Essentially all of this recent progress on lower bounds has been obtained by a technique known as *round elimination*. The technique works for a class of problems known as *locally checkable* problems [33, 16]. Informally, a locally checkable problem is a problem for which there exists a constant-time algorithm satisfying the following: if a given solution is correct, the algorithm *accepts* on all nodes; otherwise, the algorithm *rejects* on at least one node. This class encompasses many of the most fundamental problems studied in the context of the LOCAL model, such as maximal matching, maximal independent set, and different variants of graph coloring.

Round Elimination. On a very high level, round elimination works as follows. Given a problem Π provided in the proper language, the round elimination framework provides a way to mechanically construct a problem $\Pi' = \hat{\mathcal{R}}(\Pi)$ that is exactly one round easier (under some mild assumptions). That is, if Π can be solved in R rounds, then Π' can be solved in $R - 1$ rounds (and vice versa).[1] For proving an R-round lower bound on problem Π, one then has to show that the problem $\hat{\mathcal{R}}^{(R-1)}(\Pi)$, or a relaxation of it, is not trivial, i.e., it cannot be solved in 0 rounds.

In its modern form, round elimination has first been used to show that the problems of computing a sinkless edge orientation or a Δ-vertex coloring of G require $\Omega(\log \log n)$ rounds with randomization and $\Omega(\log n)$ rounds deterministically [17, 19].[2] Subsequently Brandt [16] showed that round elimination can be applied to essentially every locally checkable problem and if a problem Π is specified in the right language, the problem $\hat{\mathcal{R}}(\Pi)$ can be computed in a fully automatic way. *Automatic round elimination* in the following led to a plethora of new distributed lower bounds. We next list some of the highlights. In [3], it was shown that even in regular trees, computing a maximal matching requires $\Omega(\min\{\Delta, \log_\Delta \log n\})$ rounds with randomized algorithms and $\Omega(\min\{\Delta, \log_\Delta n\})$ rounds with deterministic algorithms. Previously, the best known lower bound as a function of Δ for this problem was only

[1] Formally, round elimination has to be performed on a weaker version of the LOCAL model, which is known as the port numbering model. In the port numbering model, nodes do not have unique IDs, but they can distinguish their neighbors through different port numbers. Round elimination lower bounds in the port numbering model can then be lifted to lower bounds in the standard LOCAL model [5].

[2] We remark that although phrased differently, the classic proofs that 3-coloring a ring requires $\Omega(\log^* n)$ rounds [32, 29] can also be seen as round elimination proofs.

$\Omega(\log \Delta / \log\log \Delta)$ [28]. By a simple reduction, the same lower bound as for maximal matching also holds for computing a maximal independent set (MIS). In later work, the same lower bound was also proven directly for the MIS problem on trees and it was generalized in particular to the problems of computing ruling sets and of computing maximal matchings in hypergraphs, leading to tight (as a function of Δ) lower bounds for those problems [8, 4, 5, 7].

We emphasize that all the recent progress on developing new lower bounds for locally checkable problems in the LOCAL model has only been possible because the work of Brandt [16] describes an *automatic and generic* way to turn any locally checkable problem (given in the right formalism) into a locally checkable problem that is exactly one round easier. Moreover, for much of the progress, it was crucial that there exists efficient software as described by Olivetti in [34] that can be used to apply round elimination to concrete locally checkable problems.

Unfortunately, while round elimination has been extremely successful for proving many new lower bounds for computing locally checkable graph problems, the method has so far not been able to provide new lower bounds for many of the standard variants of *distributed graph coloring* and thus for some of the most important and most well-studied locally checkable problems. When applying round elimination to standard $(\Delta+1)$-coloring and related graph coloring problems, the descriptions of the problems in the sequence obtained by applying $\hat{\mathcal{R}}(\cdot)$ iteratively grow doubly exponential in *each* round elimination step (i.e., with each application of $\hat{\mathcal{R}}(\cdot)$) and thus even the one round easier problem $\hat{\mathcal{R}}(\Pi)$ often becomes too complex to understand.

We are convinced that in order to continue the present success story, further developing the existing automatic techniques will be indispensible and the main objective of this paper is to provide more efficient and more powerful methods for finding distributed lower bounds in an automatic fashion.

Distributed Coloring. As a concrete application, we aim to make progress towards obtaining lower bounds for distributed coloring problems. To achieve this, we consider the problem of computing a *d-defective c-coloring*. For two parameters c and d, a d-defective c-coloring of a graph $G=(V,E)$ is a partition of V into c color classes so that every node $v \in V$ has at most d neighbors that have the same color as v. Such colorings have become an important tool in many recent distributed coloring algorithms [15, 12, 13, 11, 14, 27, 6, 10, 23]. In [23], it is also argued that further progress on defective coloring algorithms might be key towards obtaining faster distributed $(\Delta+1)$-coloring algorithms and proving hardness results on distributed defective coloring algorithms might therefore also provide insights into understanding the hardness of the standard $(\Delta+1)$-coloring problem. To obtain proper colorings, defective colorings are commonly used as a subroutine in a recursive manner and to obtain efficient coloring algorithms using few colors, it would be particularly convenient to have algorithms that efficiently compute defective colorings with c colors and defect only $(1+o(1))\Delta/c$. Such defective colorings always exist [30] and efficient distributed algorithms for computing such colorings would immediately lead to faster $O(\Delta)$-coloring algorithms and potentially also to faster $(\Delta+1)$-coloring algorithms. In fact, a generalized variant of $(1+o(1))\Delta/2$-defective 2-colorings of line graphs have recently been used in a breakthrough result that obtains the first $\operatorname{poly}\log\Delta + O(\log^* n)$-round algorithm for computing a $(2\Delta-1)$-edge coloring of a graph [6].

In contrast, the best known algorithms for computing an $O(\Delta)$ or $(\Delta+1)$-vertex coloring require time polynomial in Δ [11, 22, 14, 31]. For vertex coloring, it is already known that computing $(1+o(1))\Delta/2$-defective 2-colorings requires $\Omega(\log n)$ rounds even in bounded-

degree graphs [9]. This raises the important question whether an increased number of $c > 2$ colors can admit the desired efficient $(1 + o(1))\Delta/c$-defective c-colorings. Already the case of $c = 3$ was wide open previous to our work and an important open problem in its own right: obtaining the desired efficient defective coloring algorithm for $c = 3$ would have fundamental consequences by improving the complexity of $O(\Delta)$-coloring (and of $(\Delta + 1)$-coloring if extendable to list defective colorings [23]), while proving a substantial lower bound for any such algorithm might pave the way for proving similar lower bounds for larger c in the future. As one of the main technical results of this paper, we show that computing $(1 + o(1))\Delta/3$-defective colorings (and in fact $(1 - o(1))\Delta/2$-defective colorings) with 3 colors requires $\Omega(\log n)$ rounds. We conjecture that a similar result should also hold for more than 3 colors and we hope that such a result can be proven by extending the techniques that we introduce in this paper.

1.1 Our Contributions and High-Level Ideas of Our Techniques

In the present paper, we take the task of automating round elimination and thus automating the search for distributed lower bounds one step further. In the following, we provide a high-level discussion of the contributions of the paper.

1.1.1 An Automatic Way of Generating Round Elimination Fixed Points

Chang, Kopelowitz, and Pettie [19] showed that in the LOCAL model, every locally checkable problem Π can either be solved deterministically in $f(\Delta) \cdot O(\log^* n)$ rounds (for some function $f(\cdot)$) or Π has a deterministic $\Omega(\log_\Delta n)$ and a randomized $\Omega(\log_\Delta \log n)$ lower bounds. In the following, we call problems of the first type *easy* problems and problems of the second type *hard* problems.

Fixed Points Imply Hardness Results. A particularly elegant way to prove that a problem is of the second type is through round elimination fixed points. A locally checkable problem Π is called a round elimination fixed point if $\hat{\mathcal{R}}(\Pi) = \Pi$, i.e., if the problem that is "one round easier" than Π is Π itself. We say that a problem Π is a *non-trivial fixed point* if Π is a round elimination fixed point that cannot be solved in 0 rounds. If a problem Π is a non-trivial fixed point, existing standard techniques directly imply that Π is a hard problem, i.e., that any deterministic LOCAL algorithm to solve Π requires at least $\Omega(\log_\Delta n)$ rounds and every randomized such algorithm requires at least $\Omega(\log_\Delta \log n)$ rounds (see, e.g., [5]). Moreover, we obtain the same lower bounds for Π if Π is not a fixed point itself but can be relaxed to a non-trivial fixed point $\tilde{\Pi}$. In fact, while interesting problems exist that are non-trivial fixed points themselves (see, e.g., [17]), finding a non-trivial *fixed point relaxation* $\tilde{\Pi}$ for Π (which we may simply call a *fixed point for* Π) is a more common way to prove lower bounds for a given problem Π (see, e.g., [2, 5, 7]). Furthermore, as shown in [5, 7], surprisingly, fixed points can also be used to prove lower bounds on the Δ-dependency of *easy* problems, i.e., problems that can be solved in time $f(\Delta) \cdot O(\log^* n)$.

Fixed Points Can Be Large. In order to understand the distributed complexity of locally checkable problems, we therefore need methods to find non-trivial fixed points for such problems in case such fixed points exist. We argue that, similarly to performing and analyzing round elimination, also finding new fixed points will in many cases require some automated support for searching for fixed points. Note that in general, even for a relatively simple problem Π with a small description, the smallest fixed point relaxation $\tilde{\Pi}$ of Π might

be much more complex and have a much larger description than the original problem Π. Consider for example the Δ-coloring problem in Δ-regular graphs. While the problem itself can be described[3] with Δ different labels and Δ different node configurations (one for each possible color), the round elimination fixed point for Δ-coloring that has been described in [5] consists of 2^Δ different labels and $2^\Delta - 1$ different node configurations (and no smaller fixed point for Δ-coloring is known or suspected to exist). Finding fixed points for problems that are not as symmetric and not as well-behaved as Δ-coloring might quickly become infeasible when it has to be done by hand, even when using the support of existing software for performing single round elimination steps.

Our Contribution: a Procedure for Finding Fixed Points Automatically. While the round elimination procedure is fully automatic in the sense that it gives a mechanical way to compute $\hat{\mathcal{R}}(\Pi)$ as a function of Π, finding fixed point relaxations is a *manual* process, where it is required to *guess* what is the right relaxation $\tilde{\Pi}$ of Π and then test if $\tilde{\Pi}$ is indeed a non-trivial fixed point.

One of the major open questions in the field is understanding whether one can automatically *decide* whether a given problem is easy or hard. Essentially, the only known way to prove that a problem is hard is producing a non-trivial fixed point relaxation of it, and it is not even known if all hard problems admit a non-trivial fixed point relaxation.

As our first main contribution, we make substantial progress on this question, by providing a method to *automatically* generate relaxations $\tilde{\Pi}$ of a given locally checkable problem Π that are *guaranteed to be* fixed points under the round elimination framework. While there is no formal guarantee that the resulting problem is non-trivial (and hence that the procedure succeeded in giving a lower bound), this procedure is able to automatically derive *all* fixed point relaxations that have been manually obtained in the past. Moreover, we additionally show that the procedure provides novel results.

Our Contribution: a First Simple Application of Our Procedure. As a first direct application we get a simpler proof of a result of [5]: by applying our method to the Δ-coloring problem, we directly obtain the Δ-coloring fixed point that was presented in [5].

1.1.2 Lower Bounds for Defective Coloring Problems

Understanding the complexity of $(\Delta+1)$-coloring is one of the major open questions in the field, and one of the very few techniques known to be able to prove lower bounds for local problems is round elimination. Unfortunately, $(\Delta+1)$-coloring, and many other variants of coloring, behave very badly in round elimination, in the sense that the problem Π' constructed via round elimination is typically doubly exponentially larger than the original problem Π. Understanding *defective* colorings seems to be one of the main obstacles that we need to overcome in order to make progress on $(\Delta+1)$-coloring, for different reasons:

- While we still cannot *understand* the resulting problem Π', we can *observe* that it is some variant of coloring that includes, as a subproblem, variants of *defective* coloring. Hence, in order to prove *lower bounds* for $(\Delta+1)$-coloring, we need to understand defective colorings first;
- In [23] it has been argued that improving defective coloring algorithms might be the key for improving *upper bounds* for $(\Delta+1)$-coloring.

[3] For an introduction to the description of problems, see Section 1.1.3 or Section 3.2.

For defective 2-coloring, hardness results are known [9]. However, such results do not say much about defective c-coloring for $c \geq 3$, because defective 2-coloring is special, in the sense that this problem seems to be much harder than the case $c \geq 3$. In fact, even on graphs where each node has some minimum degree equal to some large constant, even the seemingly simpler task of requiring each node of the graph to have at least *two* neighbors of a different color is hard. While this is not our main contribution regarding defective coloring, we show that our fixed point procedure is able to automatically obtain a non-trivial fixed point relaxation for defective 2-coloring.

The problem of computing a d-defective 3-coloring is much more interesting, since, differently from defective 2-coloring, for a large range of values of d, it is known to be *easy*. In our work, we make substantial progress on defective coloring, by proving that, for $d \leq \frac{\Delta-3}{2}$, the d-defective 3-coloring problem is *hard*. Such a result is obtained by applying our fixed point procedure on defective 3-coloring. However, the obtained fixed point is so large that we needed to introduce an additional technique to let a computer verify that the resulting problem is indeed non-trivial. We believe that the three new techniques that we introduce to obtain our result on defective 3-coloring (namely, a new way of applying round elimination, a way to automatically compute fixed points, and a way to let computers verify that an entire family of problems is non-trivial) will pave the way to proving lower bounds for d-defective c-coloring for $c > 3$. In the following, we give more details on our contributions on defective coloring.

Our Contribution: Defective 2-Coloring. Not many bounds on the complexity of defective colorings are known (we discuss known bounds in the full version of this paper). An exception is the case of defective colorings with 2 colors, which is understood. By computing an MIS (which can be done in $O(\Delta + \log^* n)$ rounds [15]) and assigning the MIS nodes one of the colors and the remaining nodes the other color, one obtains a $(\Delta - 1)$-defective 2-coloring of the graph. Interestingly, the problem becomes hard if we try to just go one step further: in [9], it was shown that computing a $(\Delta - 2)$-defective 2-coloring is a hard problem. This result has been shown via a reduction from the hardness of sinkless orientation. However, this reduction is based on the construction of virtual graphs on which the defective coloring algorithm is executed in order to obtain a sinkless orientation on the original graph, and in particular the lower bounds are not proved by providing a non-trivial fixed point. As a second application of our fixed point generation method, we show the following.

There exists a non-trivial fixed point relaxation for $(\Delta - 2)$-defective 2-coloring.

This result is significant in light of the fundamental open question stated in [18, 5] asking whether, for every locally checkable problem Π that has a deterministic $\Omega(\log_\Delta n)$ and a randomized $\Omega(\log_\Delta \log n)$ lower bound, such a lower bound can be proven via a round elimination fixed point. The $(\Delta - 2)$-defective 2-coloring problem was one of an only very small number of such problems for which previously no fixed point lower bound proof was known.

Our Contribution: Defective 3-Coloring. As a main application of our automatic fixed point procedure, we study the defective coloring problem with 3 colors. From the arbdefective coloring lower bound of [5], it is known that d-defective 3-coloring is hard if $3(d+1) \leq \Delta$ and thus if $d \leq \frac{\Delta}{3} - 1$. In [9], it was further shown that if $d \geq \frac{2\Delta-4}{3}$, d-defective 3-coloring can be solved in $O(\Delta + \log^* n)$ rounds. By using our fixed point method, we manage to partially close this gap by proving the following statement.

For $d \leq \frac{\Delta-3}{2}$, the d-defective 3-coloring problem is a hard problem, i.e., it requires $\Omega(\log_\Delta n)$ rounds deterministically and $\Omega(\log_\Delta \log n)$ rounds with randomization.

This in particular implies that there is no $(1+o(1))\Delta/3$-defective 3-coloring algorithm that violates those time lower bounds, thereby ruling out the possibility of using defective 3-coloring as an approach for attacking $O(\Delta)$ and $(\Delta+1)$-coloring in the manner outlined before Section 1.1.

We note that the fixed point that we automatically generate for this problem is highly non-trivial, and that manually proving that the fixed point that we provide is indeed a fixed point would require to perform a case analysis over hundreds of cases. For this reason, we do not manually prove that the fixed point that we provide is indeed a fixed point. Instead, we provide a way to automate this process, by reducing the problem of determining whether a problem is a fixed point to the problem of proving that certain systems of inequalities have no solution. The remaining task of showing that said systems have no solution can be performed automatically via computer tools. This automatization process provides a partial answer to Open Question 9 in [5].

1.1.3 A More Efficient Method for Performing Round Elimination

Another major contribution of our work is providing a new way of applying round elimination. In particular, we provide a new procedure for computing the problem $\hat{\mathcal{R}}(\Pi)$. We would like to point out that after our work appeared online, our procedure has already proved to be extremely helpful for writing proofs based on round elimination. In fact, our procedure has been used to prove quantum advantage in the distributed setting [1]. We now provide more details on this procedure, and how we modified it to obtain the fixed point procedure mentioned in Section 1.1.1.

As a first step, in our work, we first provide a novel way for computing a locally checkable problem Π' that is exactly one round easier than Π. Then, we show that, by applying such a procedure in a slightly modified way, instead of obtaining the problem Π', we obtain some problem $\tilde{\Pi}$ which is guaranteed to be a fixed point relaxation of Π. While in some cases the obtained problem $\tilde{\Pi}$ may be solvable in 0 rounds (i.e., this *must* be the case when applying the procedure on an *easy* problem), the results presented in Section 1.1.2 are obtained by proving that the fixed points that we get by applying the procedure on defective colorings are non-trivial.

While our new procedure for applying the round elimination technique has applications for finding fixed points, this procedure is interesting on its own. In order to better explain the reason, we first highlight the main issue of the standard way of applying round elimination. While for a given locally checkable problem Π, the framework of Brandt [16] gives a fully automatic way for computing a locally checkable problem Π' that is exactly one round easier than Π, this computation is in general not computationally efficient. To illustrate why, we somewhat informally sketch how round elimination works (for a formal description we refer to Section 3.3).

How Round Elimination Works. For the automatic round elimination framework, a locally checkable problem on a Δ-regular graph $G=(V,E)$ is formalized on the bipartite graph H between the nodes V and the edges E of G.[4] That is, H is obtained by adding an additional

[4] More generally, round elimination can be defined on biregular bipartite graphs or hypergraphs (see Section 3).

node in the middle of the edges in E. Each edge of G is thus split into 2 halfedges. A solution to a locally checkable problem is given by an assignment of labels from a finite alphabet Σ to all edges of H (i.e., to each halfedge of G). The validity of a solution is given by a set of allowed node and edge configurations, where a node configuration is a multiset of labels of size Δ and an edge configuration is a multiset of labels of size 2. One step of round elimination on G is done by performing two steps of round elimination on H (note that one round on G corresponds to two rounds on H). When starting from a node-centric problem Π (i.e., a problem where the nodes in H corresponding to *nodes* in G assign the labels to their incident half-edges), the first step transforms Π into an edge-centric problem Π' that is exactly one round easier on H and the second step transforms the problem into a node-centric problem Π'' that is one round easier than Π' on H and thus one round easier than Π on G. The label set Σ' of Π' is the power set 2^Σ of Σ and the label set Σ'' of Π'' is the power set of Σ'. The allowed edge configurations of Π' are, roughly speaking, the multisets $\{\mathsf{L}_1, \mathsf{L}_2\}$ of labels $\mathsf{L}_1, \mathsf{L}_2 \in \Sigma' = 2^\Sigma$ such that *for all* $\ell_1 \in \mathsf{L}_1$ and $\ell_2 \in \mathsf{L}_2$, $\{\ell_1, \ell_2\}$ is an allowed edge configuration of Π.[5] The allowed node configurations of Π' are all the multisets $\{\mathsf{L}_1, \dots, \mathsf{L}_\Delta\}$ of labels $\mathsf{L}_i \in \Sigma'$ (that appear in some allowed edge configuration of Π') such that *there exists* an allowed node configuration $\{\ell_1, \dots, \ell_\Delta\}$ with $\ell_i \in \mathsf{L}_i$ in problem Π. In the second step, Π'' is obtained in the same way from Π', but by exchanging the roles of nodes and edges. That is, in the second step, the "for all" quantifier is applied to the allowed node configurations and the "exists" quantifier is applied to the allowed edge configurations (of Π').

The Computationally Expensive Part. Note that from a computational point of view, it is mainly the application of the "for all" quantifier on the edge side when going from Π to Π' and even more importantly on the node side when going from Π' to Π'' that is challenging. When implemented naively, one has to iterate over all possible size-2 multisets of Σ' in the first step and over all possible size-Δ multisets of Σ'' in the second step. While in general, the problem Π'' that is one round easier than the original problem Π on G can be doubly exponentially larger than Π, for interesting problems this is often not the case. For such more well-behaved problems, the "for all" case can potentially be computed in a much more efficient way.

Our Contribution. As our final contribution, we give a new elegant way to perform the application of the "for all" quantifier in round elimination. The method makes use of the fact that often the node and edge configurations of a problem can be represented by a relatively small number of *condensed configurations*. A condensed node or edge configuration is a multiset $\{S_1, \dots, S_k\}$ (where $k = \Delta$ for nodes and $k = 2$ for edges) of sets $S_1, \dots, S_k \subseteq \Sigma$ of labels, representing the set of all configurations $\{\ell_1, \dots, \ell_k\}$ for which $\ell_i \in S_i$ for all $i \in \{1, \dots, k\}$. We prove that the "for all" part of round elimination can be performed by a simple process that consists of steps of the following kind. In each step, we take two condensed configurations of the current problem and we combine those condensed configurations in some way to generate new condensed configurations. We then remove redundant configurations and continue until such a step cannot generate any new condensed configurations. In the end, each condensed configuration $\{S_1, \dots, S_k\}$ is interpreted as a multiset of labels of the new problem. We formally define the process and prove its correctness in Section 4.

[5] In the formally precise definition of the set of edge configurations of Π' provided in Section 3.3, we'll refine this definition slightly.

Informally, we prove that each configuration of the resulting problem can be described as a binary tree, where leaf nodes are condensed configurations of the original problem, and each internal node of the tree is the configuration obtained by combining its two children.

Since our new procedure is mainly used as a tool for obtaining fixed points, we do not formally state the benefits of this new procedure. However, we informally highlight the following:

- The new procedure avoids the cost of enumerating all possible size-Δ multisets of Σ'' in the second step, and its running time only depends on the number of input configurations, output configurations, and the height of the aforementioned trees. Such trees have height at most $\Delta \cdot |\Sigma'|$, and we observe that, for many natural problems, the height is much smaller. We thus obtain that, for many problems of interest, the running time of the new procedure is output-sensitive.
- Thanks to the new procedure, we obtain that, in order to check whether a problem is a fixed point, it is sufficient to check whether the combination of pairs of condensed configurations does not create new configurations. While this drastically reduces the time complexity of checking whether a problem is a fixed point, this also makes it much easier to *prove* that a problem is a fixed point. In fact, in the latter case, it is sufficient to consider two configurations at a time, instead of going through an exponential number of cases. We point out that, even if some friendly oracle gave us the fixed point for defective 3-coloring (which we present in the full version of this paper), we believe that, without exploiting this new procedure, proving that such a problem is indeed a fixed point would not have been possible.

2 Road Map

Further related work. In the full version of this paper, we provide further related work. More in detail, we discuss existing fixed points, the surprising fact that fixed points can be used to prove lower bounds as a function of Δ, and defective coloring.

Preliminaries. In Section 3, we provide some preliminaries. We first define the model of computation and the language that we use to formally describe problems. Then, we describe the round elimination framework.

A new way of applying round elimination. On a high level, round elimination allows us to start from a problem Π and to compute a problem Π' that, under some assumptions, is exactly one round easier (in the distributed setting) than Π. As it will become clear in Section 3, computing Π' as a function of Π can be a tricky process. In Section 4 we provide a novel and simplified way to compute Π' as a function of Π, and in the full version of this paper we prove its correctness.

Fixed point generation. A problem Π' is a non-trivial fixed point relaxation of Π if it satisfies the following: Π' can be solved in 0 rounds if we are given a solution for Π; Π' cannot be solved in 0 rounds in the so-called port numbering model (see Section 3 for the definition of this model); By applying round elimination on Π', we obtain Π' itself.

It is known by prior work (see Theorem 1) that, if there exists a non-trivial fixed point relaxation for a problem Π, then Π requires $\Omega(\log_\Delta n)$ rounds for deterministic algorithms and $\Omega(\log_\Delta \log n)$ rounds for randomized ones. In the full version of this paper, we provide an automatic way to obtain non-trivial fixed point relaxations. More in detail, we provide a

procedure FixedPoint that takes as input a problem Π and an object D (called diagram), and it produces a problem Π' that is always guaranteed to be a fixed point. Whether such a fixed point is non-trivial depends on Π and on the choice of D.

Selecting the right diagram. As mentioned, the choice of the diagram D may affect the triviality of the obtained fixed point. In the full version of this paper, we first provide a generic way to construct a diagram as a function of Π, that we call *default diagram*. Then, we show possible ways to modify the default diagram in the case in which the fixed-point obtained with the default diagram is a trivial one.

An alternative proof for the hardness of Δ-coloring. In the full version of this paper, we show a first application of our fixed point procedure, by providing a non-trivial fixed point relaxation for the Δ-coloring problem. Such a fixed point was already shown in [5], but here we show a much easier proof. While this section is not the main contribution of our work, its main purpose is to warm-up the reader for what comes later.

An alternative proof for the hardness of defective 2-coloring. In the full version of this paper, we show another application of our fixed point procedure, by providing a non-trivial fixed point relaxation for the $(\Delta - 2)$-defective 2-coloring problem. This is one of the few problems for which an $\Omega(\log_\Delta n)$ lower bound is known by prior work [9], but a non-trivial fixed point relaxation for this problem was unknown. Whether a non-trivial fixed point relaxation exists for all problems that require $\Omega(\log_\Delta n)$ deterministic rounds is one of the major open questions about round elimination, and hence we make progress in understanding it. Again, this result is not the main contribution of our work, and its main purpose is to prepare the reader for what comes next.

Defective 3-coloring. In the full version of this paper, we use our fixed point procedure to show a lower bound for defective 3-coloring. While the proofs for Δ-coloring and defective 2-coloring require a relatively short case analysis, the proof for defective 3-coloring requires to analyze hundreds of cases. For this reason, we prove that such a case analysis can be performed automatically by using computer tools. In particular, we reduce the task of checking whether a given problem Π is the result of applying our fixed point procedure, to proving that all systems of inequalities belonging to a certain finite set have no solution, which can be checked automatically via computer tools.

Open questions. We present some open questions in the full version of this paper.

3 Preliminaries

3.1 The LOCAL Model

The computational model that we consider is the standard LOCAL model of distributed computing [29, 35], where the nodes V of a graph $G = (V, E)$ communicate over the edges E. More precisely, time is divided into synchronous rounds, and in each round each node can send an arbitrarily large message to each neighbor. Moreover, between sending messages, nodes can perform any internal computation on the information they gathered so far. In the beginning of the computation, each node v is aware of its own degree $\deg(v)$, and has an internal ordering of its incident edges represented by the *ports* $1, \ldots, \deg(v)$ being assigned

bijectively to v's incident edges. We also assume that each node is aware of the number n of nodes and the maximum degree Δ of the input graph. As we will prove lower bounds in this work, this assumption makes our results only stronger. Moreover, each node is equipped with some *symmetry-breaking information* to avoid trivial impossibilities: in the case of deterministic algorithms, each node is assigned some globally unique ID of length $O(\log n)$ bits; in the case of randomized algorithms, each node instead has access to an unlimited amount of private random bits. Each node executes the same algorithm that governs which messages a node sends (depending on the accumulated knowledge of the node) and what the node outputs at the end of the computation. Each node has to terminate at some point and then provide a local output; all local outputs together form the global solution to the problem. The (*round* or *time*) *complexity* of a distributed algorithm is the number of rounds until the last node terminates. In the randomized setting, as usual, the algorithms are required to be Monte-Carlo algorithms that produce a correct solution with high probability, i.e., with probability at least $1 - 1/n$.

While the lower bounds we prove hold in the LOCAL model, for technical reasons we will also make use of the *port numbering* model along the way. The (deterministic) port numbering model is the same as the deterministic LOCAL model apart from two differences:

1. No symmetry-breaking information is provided, i.e., nodes are not equipped with IDs.
2. For each hyperedge e, a total order on the set of incident nodes is provided (which can be formalized via a bijection between this node set and the set $\{1, \dots, k\}$, where k denotes the number of nodes contained in e).

The second difference can be seen as an analog (on the hyperedge side) of the port numbers via which the nodes can distinguish between incident hyperedges.

3.2 Problems

The problems we study in this work fall into the class of *locally checkable problems*. Locally checkable problems are problems that can be defined via local constraints and encompass the vast majority of problems studied in the LOCAL model. A modern formalism to define these problems is given by the so-called black-white formalism that we will also use in this paper. In fact, as we will see, this formalism captures locally checkable problems not only on graphs, but more generally on hypergraphs (where we will denote the maximum number of nodes in a hyperedge by δ). In the full version of this paper we provide an example illustrating (some of) the definitions provided in this section.

The black-white formalism. In the black-white formalism, a locally checkable problem is given as a triple $\Pi = (\Sigma_\Pi, \mathcal{N}_\Pi, \mathcal{E}_\Pi)$. Here, Σ_Π is a finite set of elements, called *labels*, $\mathcal{N}_\Pi = (\mathcal{N}_1, \dots, \mathcal{N}_\Delta)$ and $\mathcal{E}_\Pi = (\mathcal{E}_1, \dots, \mathcal{E}_\delta)$, where each $\mathcal{N}_i$ and $\mathcal{E}_i$ is a collection of multisets of cardinality i with labels from Σ_Π. We call $\mathcal{N}_\Pi$ the *node constraint* of Π and $\mathcal{E}_\Pi$ the *edge constraint* of Π. On a hypergraph, a correct solution for Π is an assignment of labels from Σ_Π to the incident node-hyperedge pairs such that for each node v, the multiset of labels corresponding to v is contained in $\mathcal{N}_{\deg(v)}$, and analogously for hyperedges w.r.t. the respective $\mathcal{E}_i$. More formally, let $\mathcal{F}$ denote the set of pairs (v, e) where e is a hyperedge incident to v. A correct solution for Π on a hypergraph $G = (V, E)$ is a mapping $f \colon \mathcal{F} \to \Sigma_\Pi$ such that, for each $v \in V$, we have $\{f(v, e') \mid e' \ni v\} \in \mathcal{N}_{\deg(v)}$, and, for each $e \in E$, we have $\{f(v', e) \mid v' \in e\} \in \mathcal{E}_{\mathrm{rank}(e)}$. Here, the *rank* $\mathrm{rank}(e)$ of a hyperedge e is the number of nodes contained in e, and the displayed sets are to be understood as multisets.

When solving a locally checkable problem in the distributed setting, each node v has to output one label for each "incident" node-hyperedge pair in $\mathcal{F}$ such that the induced global solution is correct. While the improvements for the general round elimination technique

(discussed below) that we will obtain in this work apply to the general hypergraph setting, for the results about concrete problems that we provide we can restrict attention to the special case of graphs. In this special case, each hyperedge is of rank 2, and consequently we will replace the edge constraint $(\mathcal{E}_1, \ldots, \mathcal{E}_\delta)$ by $\mathcal{E}_2$. Moreover, to simplify notation, in this case, we will set $\mathcal{E} := \mathcal{E}_2$.

We remark that besides providing a formalism for graphs by considering them as a special case of hypergraphs, the black-white formalism provides a (different) way to encode and study problems on *bipartite* graphs, by identifying the "black" nodes in the bipartition with the *nodes* in the above formalism, and the "white" nodes with the *hyperedges*. This relation to bipartite graphs is also where the name "black-white formalism" comes from.

As can be observed, the definition of the problems in this formalism depends on Δ (and δ), which provides the power to also describe important problems like $(\Delta+1)$-coloring in this formalism. If we are to be very precise, in this formalism each problem is a collection of problems indexed by Δ (and, if considered on hypergraphs, δ). Throughout the paper, we implicitly assume that some (arbitrary) Δ (and, if required, some δ) is fixed. Note that this does not impact the generality of our results.

Finally, we remark that, for simplicity, we consider two locally checkable problems given in the black-white formalism as identical if one can be obtained from the other by renaming the labels used to describe the latter.

Configurations. We will use the term *configuration* to refer to a multiset of labels, and write it in either of the two equivalent forms $\{\ell_1, \ldots, \ell_i\}$ and $\ell_1 \ldots \ell_i$. Note that the order of the ℓ_j does not matter (also in the second form): all configurations that can be obtained from a configuration by reordering are considered to be the same configuration. When referring to the multiset of labels assigned to the pairs (v, e') incident to a fixed node v, we will use the term *node configuration*; when referring to the multiset of labels assigned to the pairs (v', e) corresponding to a fixed (hyper)edge e, we will use the term *edge configuration*. Moreover, for simplicity we may slightly abuse notation by writing $\{\ell_1, \ldots, \ell_i\} \in \mathsf{L}_1 \times \cdots \times \mathsf{L}_i$ if $\mathsf{L}_1, \ldots, \mathsf{L}_i$ are sets containing the labels $\ell_1, \ldots, \ell_i$, respectively.

It will be convenient to refer to certain collections of configurations in a condensed manner. A *condensed configuration* $\mathcal{C}$ is a configuration $\{\mathsf{L}_1, \ldots, \mathsf{L}_i\}$ of sets of labels. Configuration $\mathcal{C}$ is to be understood as the set of all configurations $\{\ell_1, \ldots, \ell_i\} \in \mathsf{L}_1 \times \cdots \times \mathsf{L}_i$ (though we will also consider the condensed configuration $\mathcal{C}$ as a configuration of sets when convenient). To indicate that a configuration of sets represents a condensed configuration, we will often write each set in the configuration in the form $[\ell_1 \ \ldots \ \ell_j]$ (unless the set only contains one element ℓ, in which case we will simply write the set as ℓ).

Diagrams. A useful way of capturing certain aspects of problems is via so-called diagrams. A *diagram* $D = (\Sigma_D, E_D)$ is nothing else than a directed acyclic graph with node set Σ_D and edge set E_D. The *edge diagram* of a problem $\Pi = (\Sigma_\Pi, \mathcal{N}_\Pi, \mathcal{E}_\Pi)$ is the diagram D obtained by setting $\Sigma_D := \Sigma_\Pi$ and defining E_D as the set of those directed edges (ℓ, ℓ') that satisfy that $\ell' \neq \ell$ and, for every configuration $\{\ell_1, \ldots, \ell_\delta\} \in \mathcal{E}_\Pi$ with $\ell_i = \ell$ for some $1 \leq i \leq \delta$, also $\{\ell_1, \ldots, \ell_{i-1}, \ell', \ell_{i+1}, \ldots, \ell_\delta\} \in \mathcal{E}_\Pi$. When displaying a diagram, we often omit arrows that can be obtained as the composition of displayed arrows. We call a subset $S \subseteq \Sigma_D$ *right-closed* (w.r.t. D) if, for any edge $(\ell, \ell') \in E_D$, $\ell \in S$ implies $\ell' \in S$.

3.3 The Round Elimination Technique

In this section, we give a formal introduction to round elimination. As some of the definitions provided in this section are fairly technical, the reader is encouraged to consult the illustrating example provided in the full version of this paper alongside reading the definitions.

For technical reasons, round elimination requires the considered input (hyper)graphs to be regular (and uniform). As such, we will assume throughout the paper that every node of the input (hyper)graph has the same degree Δ and every (hyper)edge has the same rank δ (which, in the case of graphs, is simply 2). This also simplifies the representation of locally checkable problems $\Pi = (\Sigma_\Pi, \mathcal{N}_\Pi, \mathcal{E}_\Pi)$: now we can assume that $\mathcal{N}_\Pi$ and $\mathcal{E}_\Pi$ are collections of multisets of cardinalities Δ and δ, respectively, instead of sequences of similar collections. Note that, as we will prove lower bounds in this work, the inherent restriction to regular graphs makes our results only stronger.

$\mathcal{R}(\cdot)$ and $\overline{\mathcal{R}}(\cdot)$. At the heart of the round elimination technique lie the round elimination operators $\mathcal{R}$ and $\overline{\mathcal{R}}$, which are functions that take a locally checkable problem in the black-white formalism as input and return such a problem. More precisely, for a locally checkable problem $\Pi = (\Sigma_\Pi, \mathcal{N}_\Pi, \mathcal{E}_\Pi)$, the locally checkable problem $\mathcal{R}(\Pi) = (\Sigma_{\mathcal{R}(\Pi)}, \mathcal{N}_{\mathcal{R}(\Pi)}, \mathcal{E}_{\mathcal{R}(\Pi)})$ is defined as follows.

The label set $\Sigma_{\mathcal{R}(\Pi)}$ of $\mathcal{R}(\Pi)$ is simply the set of non-empty subsets of Σ_Π, i.e., $\Sigma_{\mathcal{R}(\Pi)} := 2^{\Sigma_\Pi} \setminus \{\{\}\}$. For the definition of the edge constraint $\mathcal{E}_{\mathcal{R}(\Pi)}$ of $\mathcal{R}(\Pi)$, we need the notion of a *maximal configuration*. Let $\mathcal{Z}$ be a collection of configurations of sets of labels. Then, a configuration $\mathsf{L}_1 \dots \mathsf{L}_i \in \mathcal{Z}$ is maximal (in $\mathcal{Z}$) if there is no configuration $\mathsf{L}'_1 \dots \mathsf{L}'_i \in \mathcal{Z}$ (of the same length) such that there exists a bijection $\phi\colon \{1, \dots, i\} \to \{1, \dots, i\}$ satisfying $\mathsf{L}_j \subseteq \mathsf{L}'_{\phi(j)}$ for all $1 \le j \le i$ and $\mathsf{L}_j \subsetneq \mathsf{L}'_{\phi(j)}$ for at least one $1 \le j \le i$. In other words, a configuration of sets is maximal if no other configuration in the considered configuration space can be reached by enlarging (some of) the sets (and reordering the sets).

Now we can define $\mathcal{E}_{\mathcal{R}(\Pi)}$ as follows. Let $\mathcal{E}$ denote the collection of all configurations $\mathsf{L}_1 \dots \mathsf{L}_\delta$ such that $\mathsf{L}_1, \dots, \mathsf{L}_\delta \in \Sigma_{\mathcal{R}(\Pi)}$ and for all choices $(\ell_1, \dots, \ell_\delta) \in \mathsf{L}_1 \times \dots \times \mathsf{L}_\delta$ of labels we have $\{\ell_1, \dots, \ell_\delta\} \in \mathcal{E}_\Pi$. Then, $\mathcal{E}_{\mathcal{R}(\Pi)}$ is obtained from $\mathcal{E}$ by removing all configurations that are not *maximal* in $\mathcal{E}$. Finally, the node constraint $\mathcal{N}_{\mathcal{R}(\Pi)}$ of $\mathcal{R}(\Pi)$ is defined as the collection of all configurations $\mathsf{L}_1 \dots \mathsf{L}_\Delta$ such that each L_i appears in at least one configuration from $\mathcal{E}_{\mathcal{R}(\Pi)}$ and there exists a choice $(\ell_1, \dots, \ell_\Delta) \in \mathsf{L}_1 \times \dots \times \mathsf{L}_\Delta$ of labels satisfying $\{\ell_1, \dots, \ell_\Delta\} \in \mathcal{N}_\Pi$.

The problem $\overline{\mathcal{R}}(\Pi) = (\Sigma_{\overline{\mathcal{R}}(\Pi)}, \mathcal{N}_{\overline{\mathcal{R}}(\Pi)}, \mathcal{E}_{\overline{\mathcal{R}}(\Pi)})$ is defined dually to $\mathcal{R}(\Pi)$, where the role of nodes and hyperedges are reversed. More precisely, we have the following. As before, $\Sigma_{\overline{\mathcal{R}}(\Pi)} = \Sigma_{\mathcal{R}(\Pi)} = 2^{\Sigma_\Pi} \setminus \{\{\}\}$. The node constraint $\mathcal{N}_{\overline{\mathcal{R}}(\Pi)}$ of $\overline{\mathcal{R}}(\Pi)$ is the collection of *maximal* configurations $\mathsf{L}_1 \dots \mathsf{L}_\Delta$ such that $\mathsf{L}_1, \dots, \mathsf{L}_\Delta \in \Sigma_{\overline{\mathcal{R}}(\Pi)}$ and for all choices $(\ell_1, \dots, \ell_\Delta) \in \mathsf{L}_1 \times \dots \times \mathsf{L}_\Delta$ of labels we have $\{\ell_1, \dots, \ell_\Delta\} \in \mathcal{N}_\Pi$. The edge constraint $\mathcal{E}_{\overline{\mathcal{R}}(\Pi)}$ of $\overline{\mathcal{R}}(\Pi)$ is the collection of all configurations $\mathsf{L}_1 \dots \mathsf{L}_\delta$ such that each L_i appears in at least one configuration from $\mathcal{N}_{\overline{\mathcal{R}}(\Pi)}$ and there exists a choice $(\ell_1, \dots, \ell_\delta) \in \mathsf{L}_1 \times \dots \times \mathsf{L}_\delta$ of labels satisfying $\{\ell_1, \dots, \ell_\delta\} \in \mathcal{E}_\Pi$.

We will refer to the operation of deriving $\mathcal{E}_{\mathcal{R}(\Pi)}$ from $\mathcal{E}_\Pi$ (and $\mathcal{N}_{\overline{\mathcal{R}}(\Pi)}$ from $\mathcal{N}_\Pi$) as *applying the universal quantifier (to $\mathcal{E}_\Pi$ and $\mathcal{N}_\Pi$, respectively)* and say that a problem *satisfies the universal quantifier* if it is the result of such an operation.

The hard part in computing $\mathcal{R}(\Pi)$ and $\overline{\mathcal{R}}(\Pi)$ is applying the universal quantifier. In fact, consider the problem $\mathcal{R}(\Pi)$. There is an easy way to compute $\mathcal{N}_{\mathcal{R}(\Pi)}$, that is the following. Start from all the configurations in $\mathcal{N}_\Pi$, and for each configuration add to $\mathcal{N}_{\mathcal{R}(\Pi)}$ the condensed configuration obtained by replacing each label ℓ by the set that contains all label sets in $\Sigma_{\mathcal{R}(\Pi)}$ containing ℓ.

The round elimination sequence. In the round elimination framework, the two operators $\mathcal{R}$ and $\overline{\mathcal{R}}$ are used to define a sequence of problems that is essential for obtaining complexity lower bounds via round elimination. This sequence $\Pi_0, \Pi_1, \Pi_2, \ldots$ is defined via $\Pi_{i+1} := \overline{\mathcal{R}}(\mathcal{R}(\Pi_i))$ for all $i \geq 0$, where Π_0 is the given problem of interest. The following theorem provides a way to obtain lower bounds for the complexity of Π_0 via analyzing the 0-round-solvability of the problems in the sequence. It is a simplified version of Theorem 7.1 from [5].

▶ **Theorem 1.** *Let $\Pi_0, \Pi_1, \ldots, \Pi_t$ be a sequence of problems satisfying $\Pi_{i+1} = \overline{\mathcal{R}}(\mathcal{R}(\Pi_i))$ for all $0 \leq i \leq t-1$. Moreover, let B be an integer (that may depend on n and/or Δ) such that $|\Sigma_{\Pi_i}| \leq B$ for all $0 \leq i \leq t$, and $|\Sigma_{\mathcal{R}(\Pi_i)}| \leq B$ for all $0 \leq i \leq t-1$. Then, if Π_t is not 0-round-solvable in the port numbering model, Π_0 has lower bounds of $\Omega(\min\{t, \log_\Delta n - \log_\Delta \log B\})$ rounds in the deterministic* LOCAL *model and $\Omega(\min\{t, \log_\Delta \log n - \log_\Delta \log B\})$ rounds in the randomized* LOCAL *model.*

Fixed points. As implied by Theorem 1, it is crucial for proving lower bounds via round elimination to be able to determine the 0-round solvability of problems in the round elimination sequence produced by the studied problem Π_0. A class of problems that produces very simple sequences are so-called fixed points. A locally checkable problem Π is called a *fixed point* if $\overline{\mathcal{R}}(\mathcal{R}(\Pi)) = \Pi$. Moreover, for a fixed point Π, the problem $\Pi' := \mathcal{R}(\Pi)$ is called the *intermediate problem*. Note that such an intermediate problem Π' satisfies $\mathcal{R}(\overline{\mathcal{R}}(\Pi')) = \Pi'$. We get the following corollary from Theorem 1.

▶ **Corollary 2.** *Let Π be a fixed point in the round elimination framework. Then, if Π is not 0-round-solvable in the port numbering model, Π has lower bounds of $\Omega(\log_\Delta n)$ rounds in the deterministic* LOCAL *model and $\Omega(\log_\Delta \log n)$ rounds in the randomized* LOCAL *model.*

0-round-solvability. Due to Theorem 1, we are interested in determining whether a problem can be solved in 0 rounds or not. For technical reasons, throughout the paper, whenever we consider the 0-round-solvability of a problem, we will consider it in the *port numbering model*. In the port numbering model, 0-round-solvability admits a simple characterization: a problem Π is 0-round-solvable if and only if there is a configuration $\ell_1 \ \ldots \ \ell_\Delta \in \mathcal{N}_\Pi$ such that, for any δ (not necessarily distinct) labels $\ell'_1, \ldots, \ell'_\delta \in \{\ell_1, \ldots, \ell_\Delta\}$, it holds that $\ell'_1 \ \ldots \ \ell'_\delta \in \mathcal{E}_\Pi$. We will use the terms *trivial* and *non-trivial* to refer to 0-round-solvable and non-0-round-solvable problems, respectively. In particular, we will be interested in trivial and non-trivial fixed points.

For the interested reader, in the full version of this paper we provide an example of the application of the round elimination procedure to a simple problem called sinkless orientation.

4 A New Way of Applying Round Elimination

In this section, we describe a novel and simple way for applying the round elimination technique. As already discussed in Section 3.3, the hard and error-prone part in applying the $\mathcal{R}(\cdot)$ and $\overline{\mathcal{R}}(\cdot)$ operators consists in applying the universal quantifier. Let $\Pi = (\Sigma_\Pi, \mathcal{N}_\Pi, \mathcal{E}_\Pi)$ be the problem of interest, where $\mathcal{N}_\Pi$ contains multisets of size Δ and $\mathcal{E}_\Pi$ contains multisets of size δ. Also, let $\Pi' = \mathcal{R}(\Pi) = (\Sigma_{\Pi'}, \mathcal{N}_{\Pi'}, \mathcal{E}_{\Pi'})$. Recall that applying the universal quantifier means computing $\mathcal{E}_{\Pi'}$ as follows. First, let $\mathfrak{C}$ be the maximal set such that for all $\mathsf{L}_1 \ \ldots \ \mathsf{L}_\delta \in \mathfrak{C}$ it holds that, for all i, $\mathsf{L}_i \in 2^{\Sigma_\Pi} \setminus \{\{\}\}$, and all multisets $\{\ell_1, \ldots, \ell_\delta\} \in \mathsf{L}_1 \times \ldots \times \mathsf{L}_\delta$ are in $\mathcal{E}_\Pi$. Then, $\mathcal{E}_{\Pi'}$ is obtained by removing all non-maximal configurations from $\mathfrak{C}$. This definition, if implemented in a naive manner, requires considering all possible configurations from labels in 2^{Σ_Π}, and then, for each of them, checking if all possible configurations obtained by selecting one label from each set in the configuration are contained in $\mathcal{E}_\Pi$.

4.1 A new way to compute $\mathcal{E}_{\Pi'}$

We show a drastically simplified way of applying the universal quantifier, that, at each point in time, requires to consider only two configurations and to perform elementary operations on those.

Input of the new procedure. While, formally, the given constraint $\mathcal{E}_\Pi$ is described as a set of multisets, in some cases the given constraint is described in a more compact form, that is, by providing *condensed configurations*. The procedure that we describe does not need to unpack condensed configurations into a set of non-condensed ones, and this feature allows to apply this new procedure more easily. For this reason, we assume that $\mathcal{E}_\Pi$ is described as a set Γ_Π of condensed configurations, that is, Γ_Π contains multisets, where each multiset $\mathcal{L} \in \Gamma_\Pi$ is of the form $\{\mathsf{L}_1, \ldots, \mathsf{L}_\delta\}$, and for all $1 \leq i \leq \delta$ it holds that $\mathsf{L}_i \subseteq \Sigma_\Pi$. Clearly, if we are given $\mathcal{E}_\Pi$ as a list of non-condensed configurations, we can convert it into this form by replacing each label with a singleton set. While we assume that the input is described as a set of condensed configurations, the output of the procedure is going to be a set of non-condensed configurations. We call the condensed configurations in Γ_Π *input configurations*.

Combining configurations. At the heart of our procedure lies an operation that *combines* two given configurations of sets. We now formally define what it means to combine two such configurations. Let $\mathcal{L} = \{\mathsf{L}_1, \ldots, \mathsf{L}_\delta\}$ and $\mathcal{L}' = \{\mathsf{L}'_1, \ldots, \mathsf{L}'_\delta\}$ be two configurations, where L_i and L'_i are sets. Let $\phi\colon \{1, \ldots, \delta\} \to \{1, \ldots, \delta\}$ be a bijection, i.e., a permutation of $\{1, \ldots, \delta\}$. Let $u \in \{1, \ldots, \delta\}$. Combining $\mathcal{L}$ and $\mathcal{L}'$ w.r.t. ϕ and u means constructing the configuration $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_\delta\}$ where $\mathsf{C}_i = \mathsf{L}_i \cup \mathsf{L}'_{\phi(i)}$ if $i = u$ and $\mathsf{C}_i = \mathsf{L}_i \cap \mathsf{L}'_{\phi(i)}$ otherwise. In other words, we consider an arbitrary perfect matching between the sets of the two configurations, and we take the union for one matched pair and the intersection for the remaining matched pairs. In Figure 1, we show an example of a combination of two configurations.

The New Procedure. In the following, we construct a sequence (Ψ_i) of sets of configurations until certain desirable properties are obtained. The first step of the procedure is setting $\Psi_0 = \Gamma_\Pi$. The next step is to apply a subroutine that creates Ψ_{i+1} as a function of Ψ_i, and this subroutine is repeatedly applied until we get that $\Psi_{i+1} = \Psi_i$. Let the final result be $\mathcal{E}^*_{\Pi'}$.

The subroutine computes all possible combinations of pairs of configurations (including a configuration with itself) that are in Ψ_i, for all possible permutations ϕ and for all possible choices of u. If a resulting configuration contains an empty set, the configuration is discarded. Let Ψ_{i+1} be the set of configurations obtained by starting from the configurations in Ψ_i, adding the newly computed configurations, and then removing the non-maximal ones. We call the defined procedure NewRE, which is described more formally in Algorithm 1.

In the full version of this paper, we will first provide an example of execution of our new procedure, and then we will prove that the constraint $\mathcal{E}^*_{\Pi'}$ returned by NewRE is equal to the constraint $\mathcal{E}_{\Pi'}$ as defined according to the definition of round elimination given in Section 3. Finally, we will also prove that the procedure always terminates.

$$\{\mathsf{I},\mathsf{O}\} \cup \{\mathsf{O}\} = \{\mathsf{I},\mathsf{O}\}, \qquad \{\mathsf{I},\mathsf{O}\} \cap \{\mathsf{I},\mathsf{O}\} = \{\mathsf{I},\mathsf{O}\}, \qquad \{\mathsf{O}\} \cap \{\mathsf{I},\mathsf{O}\} = \{\mathsf{O}\}$$

Figure 1 One possible way to combine $\{\mathsf{I},\mathsf{O}\}\,\{\mathsf{I},\mathsf{O}\}\,\{\mathsf{O}\}$ with itself. The resulting configuration is $\{\mathsf{I},\mathsf{O}\}\,\{\mathsf{I},\mathsf{O}\}\,\{\mathsf{O}\}$.

Algorithm 1 The new procedure.

▷ *Applies the procedure to the input configurations* Γ ◁
procedure NewRe(Γ, δ)
  $\Psi_0 \leftarrow \Gamma$
  for $i \leftarrow 0, 1, 2, \ldots$ **do**
    $\Psi \leftarrow \Psi_i$
    for all $\mathcal{L} \in \Psi_i$ **do**
      for all $\mathcal{L}' \in \Psi_i$ **do**
        for all permutations ϕ over the integers $\{1, \ldots, \delta\}$ **do**
          for all $1 \leq u \leq \delta$ **do**
            $\mathcal{C} \leftarrow$ Combine($\mathcal{L}$, $\mathcal{L}'$, δ, ϕ, u)
            if $\{\} \notin \mathcal{C}$ **then**
              $\Psi \leftarrow \Psi \cup \{\mathcal{C}\}$
    $\Psi_{i+1} \leftarrow$ DiscardNonMaximal(Ψ)
    if $\Psi_{i+1} = \Psi_i$ **then**
      break
  return Ψ_i
▷ *Combines two configurations w.r.t. a given permutation* ϕ *and position* u ◁
procedure Combine($\mathcal{L} = \{\mathsf{L}_1, \ldots, \mathsf{L}_\delta\}$, $\mathcal{L}' = \{\mathsf{L}'_1, \ldots, \mathsf{L}'_\delta\}$, δ, ϕ, u)
  for $i \leftarrow 1, \ldots, \delta$ **do**
    if $i = u$ **then**
      $\mathsf{C}_i = \mathsf{L}_i \cup \mathsf{L}'_{\phi(i)}$
    else
      $\mathsf{C}_i = \mathsf{L}_i \cap \mathsf{L}'_{\phi(i)}$
  $\mathcal{C} \leftarrow \{\mathsf{C}_1, \ldots, \mathsf{C}_\delta\}$
  return $\mathcal{C}$
▷ *Returns the set of maximal configurations of* Ψ ◁
procedure DiscardNonMaximal(Ψ)
  $S \leftarrow \{\}$
  for all $\mathcal{L} \in \Psi$ **do**
    if $\neg(\exists \mathcal{L}' \in \Psi$ s.t. $\mathcal{L}' \neq \mathcal{L}$ and Dominates($\mathcal{L}'$, $\mathcal{L}))$ **then**
      $S \leftarrow S \cup \{\mathcal{L}\}$
  return S
▷ *Determines whether* $\mathcal{L}'$ *dominates* $\mathcal{L}$ ◁
procedure Dominates($\mathcal{L}' = \{\mathsf{L}'_1, \ldots, \mathsf{L}'_\delta\}$, $\mathcal{L} = \{\mathsf{L}_1, \ldots, \mathsf{L}_\delta\}$)
  return $\exists$ permutation ϕ such that, for all $1 \leq i \leq \delta$, $\mathsf{L}_i \subseteq \mathsf{L}'_{\phi(i)}$

References

1 Alkida Balliu, Sebastian Brandt, Xavier Coiteux-Roy, Francesco D'Amore, Massimo Equi, Francois Le Gall, Henrik Lievonen, Augusto Modanese, Dennis Olivetti, Marc-Olivier Renou, Jukka Suomela, Lucas Tendick, and Isadora Veeren. Distributed quantum advantage for local problems. In *Proceedings of the 57th Annual ACM SIGACT Symposium on Theory of Computing, (STOC 2025)*, 2025.

2 Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. In *Proc. 34th Symp. on Distributed Computing (DISC)*, 2020.

3 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *Proc. 60th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 481–497, 2019. doi:10.1109/FOCS.2019.00037.

4 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Improved distributed lower bounds for MIS and bounded (out-)degree dominating sets in trees. In *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC)*, 2021. doi:10.1145/3465084.3467901.

5 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ-coloring plays hide-and-seek. In *54th ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 464–477, 2022.

6 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time polylogarithmic in Δ. In *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 15–25. ACM, 2022. doi:10.1145/3519270.3538440.

7 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed maximal matching and maximal independent set on hypergraphs. In *Proc. 34th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2632–2676, 2023. doi:10.1137/1.9781611977554.CH100.

8 Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. In *Proc. 61st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 365–376, 2020. doi:10.1109/FOCS46700.2020.00042.

9 Alkida Balliu, Juho Hirvonen, Christoph Lenzen, Dennis Olivetti, and Jukka Suomela. Locality of not-so-weak coloring. In *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2019. doi:10.1007/978-3-030-24922-9_3.

10 Alkida Balliu, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time quasi-polylogarithmic in delta. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 289–298, 2020. doi:10.1145/3382734.3405710.

11 Leonid Barenboim. Deterministic (Δ+1)-Coloring in Sublinear (in Δ) Time in Static, Dynamic, and Faulty Networks. *Journal of ACM*, 63(5):1–22, 2016. doi:10.1145/2979675.

12 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Comput.*, 22:363–379, 2010. doi:10.1007/s00446-009-0088-2.

13 Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of ACM*, 58:23:1–23:25, 2011. doi:10.1145/2027216.2027221.

14 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta + 1)$-coloring below Szegedy-Vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *Proc. 37th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 437–446, 2018. doi:10.1145/3212734.3212769.

15 Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed (Δ+1)-coloring in linear (in Δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2014. doi:10.1137/12088848X.

16 Sebastian Brandt. An automatic speedup theorem for distributed problems. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 379–388, 2019. doi:10.1145/3293611.3331611.

17 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed Lovász local lemma. In *Proceedings of the 48th ACM Symposium on Theory of Computing (STOC 2016)*, pages 479–488. ACM Press, 2016. doi:10.1145/2897518.2897570.

18 Sebastian Brandt and Dennis Olivetti. Truly tight-in-Δ bounds for bipartite maximal matching and variants. In *Proc. 39th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 69–78, 2020. doi:10.1145/3382734.3405745.

19 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM Journal on Computing*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

20 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed (Δ+1)-coloring algorithm? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, (STOC 2018)*, pages 445–456, 2018. doi:10.1145/3188745.3188964.

21 Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhon. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proc. 34th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 4409–4447, 2023. doi:10.1137/1.9781611977554.CH168.

22 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. 57th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 625–634, 2016. doi:10.1109/FOCS.2016.73.

23 Marc Fuchs and Fabian Kuhn. List defective colorings: Distributed algorithms and applications. In Rotem Oshman, editor, *Proc. 37th Int. Symp. on Distributed Computing (DISC)*, volume 281 of *LIPIcs*, pages 22:1–22:23, 2023. doi:10.4230/LIPICS.DISC.2023.22.

24 Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2148–2179. IEEE, 2024. doi:10.1109/FOCS61266.2024.00007.

25 Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhon. Improved distributed network decomposition, hitting sets, and spanners, via derandomization. In *Proc. 34th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 2532–2566, 2023. doi:10.1137/1.9781611977554.CH97.

26 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *Proc. 59th Symp. on Foundations of Computer Science (FOCS)*, pages 662–673, 2018. doi:10.1109/FOCS.2018.00069.

27 Fabian Kuhn. Faster deterministic distributed coloring through recursive list coloring. In *Proc. 32st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1244–1259, 2020. doi:10.1137/1.9781611975994.76.

28 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of ACM*, 63:17:1–17:44, 2016. doi:10.1145/2742012.

29 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

30 L. Lovász. On decompositions of graphs. *Studia Sci. Math. Hungar.*, 1:237–238, 1966.

31 Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.16.

32 Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM Journal on Discrete Mathematics*, 4(3):409–412, 1991. doi:10.1137/0404036.

33 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

34 Dennis Olivetti. Round Eliminator: a tool for automatic speedup simulation, 2019. URL: https://github.com/olidennis/round-eliminator.

35 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

36 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, 2020. doi:10.1145/3357713.3384298.

Energy-Efficient Maximal Independent Sets in Radio Networks

Dominick Banasik
Rochester Institute of Technology, NY, USA

Varsha Dani
Rochester Institute of Technology, NY, USA

Fabien Dufoulon
Lancaster University, UK

Aayush Gupta
University of Houston, TX, USA

Thomas P. Hayes
University at Buffalo, NY, USA

Gopal Pandurangan
University of Houston, TX, USA

Abstract

The maximal independent set (MIS) is one of the most fundamental problems in distributed computing, and it has been studied intensively for over four decades. This paper focuses on the MIS problem in the *radio network* model, a standard model widely used to model *wireless* networks, particularly ad hoc wireless and sensor networks. Energy is a premium resource in these networks, which are typically battery-powered. Hence, designing distributed algorithms that use as little energy as possible is crucial. We use the well-established energy model where a node can be *sleeping* or *awake* in a round, and only the awake rounds (when it can send or listen) determine the *energy complexity* of the algorithm, which we want to minimize.

We present new, more energy-efficient MIS algorithms in radio networks with *arbitrary* and unknown graph topology. We present algorithms for two popular variants of the radio model – with collision detection (CD) and without collision detection (no-CD). Specifically, we obtain the following results:

1. *CD model*: We present a randomized distributed MIS algorithm with energy complexity $O(\log n)$, round complexity $O(\log^2 n)$, and failure probability $1/\operatorname{poly}(n)$, where n is the network size. We show that our energy complexity is optimal by showing a matching $\Omega(\log n)$ lower bound.
2. *no-CD model*: In the more challenging no-CD model, we present a randomized distributed MIS algorithm with energy complexity $O(\log^2 n \log\log n)$, round complexity $O(\log^3 n \log \Delta)$, and failure probability $1/\operatorname{poly}(n)$. The energy complexity of our algorithm is significantly lower than the round (and energy) complexity of $O(\log^3 n)$ of the best known distributed MIS algorithm of Davies [PODC 2023] for arbitrary graph topology.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases Distributed Computing, Energy Complexity, Sleeping Model, Radio Networks, Maximal Independent Set

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.14

Related Version *Full Version*: https://arxiv.org/abs/2510.08244

Funding *Gopal Pandurangan*: Supported in part by NSF grant CCF-2402837 and ARO grant W911NF-231-0191.



© Dominick Banasik, Varsha Dani, Fabien Dufoulon, Aayush Gupta, Thomas P. Hayes, and Gopal Pandurangan;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The Maximal Independent Set (MIS) problem is a fundamental problem in graph theory and distributed computing, with numerous applications in network design, resource allocation, and parallel computing. It is also one of the best-studied symmetry-breaking problems in distributed networks. In particular, solving the MIS problem efficiently is crucial in radio networks due to the inherent challenges posed by *wireless* communication, such as contention and collision. For example, in the popular ad hoc wireless and sensor networks, nodes are deployed with no infrastructure; in fact, nodes may not even know which nodes are close (i.e., neighbors). Unlike wired networks, nodes cannot broadcast at will to discover their neighbors; radio interference and collisions make it unlikely for such uncoordinated communications to reliably transmit any information. To coordinate communication, one can first construct an MIS, then use it as a building block for setting up a *communication backbone*. Such communication applications of MIS have been studied extensively in ad hoc wireless and sensor networks (see e.g., [37]). However, this leads to the problem of *first constructing an MIS* starting with no underlying knowledge of the neighborhood or topology in radio networks, which is the focus of this paper.

Another major issue in radio networks, such as ad hoc sensor networks, is that nodes are typically battery-powered and hence energy-constrained. Most of the energy consumption of the nodes is when they are transmitting or listening. On the other hand, very little energy is consumed when the nodes are *sleeping*, i.e., when the radio devices are switched off; in such a state, a node does not send or listen (but messages sent to it are lost) [12, 13]. This necessitates the design of *energy-efficient* distributed algorithms where nodes try to minimize the number of rounds they send or listen in.

In this paper, we focus on designing *energy-efficient* distributed algorithms for MIS in radio networks with *arbitrary (and unknown)* topology. Distributed algorithms (and lower bounds) for constructing MIS in radio networks have been studied extensively for many years (cf. Section 1.4). Almost all of the algorithms studied, except the recent algorithm of Davies [18], assumed the underlying graph to be of a particular type, such as unit disk [34], growth-bounded [36], or more generally, bounded independence [16, 17]. Furthermore, all these works focused on minimizing the *round complexity*. In particular, the work of Davies [18] – which is most relevant to this work – gave an $O(\log^3 n)$ round distributed MIS algorithm for radio networks that works for an arbitrary (and unknown) underlying topology (throughout, n denotes the number of nodes in the network). This is the best known round-efficient MIS algorithm for radio networks that works under an arbitrary topology. However, none of the above prior works focused on designing energy-efficient algorithms in radio networks, and theier energy complexities can be *as high* as their respective round complexities.

1.1 Radio Network Model and Energy Complexity

Radio networks are characterized by their broadcast communication model, where nodes communicate by transmitting messages over shared channels. In this model, a message sent by a node can be received by all its neighbors within the transmission range. However, if multiple nodes transmit simultaneously, *collisions* occur, leading to communication failures. This necessitates the development of robust distributed algorithms that can effectively handle such collisions and ensure reliable communication.

We assume we have an underlying communication network modeled by an *arbitrary* graph G. Each node in G has a transmitter and receiver to communicate with other nodes. There is an edge between two nodes in G if they are within each other's transmission range. We

note that the graph G is *unknown* to all the nodes. In particular, we will assume that nodes do not even know who their neighbors are in the graph until they have explicitly heard from them during the algorithm. This is sometimes called the *ad hoc* model [18].

We assume that the nodes start out with shared knowledge of a value n, which is an upper bound on the number of nodes in the network; our bounds require only an estimate of n within a polynomial factor.[1] Similarly, we shall assume that the nodes have shared knowledge of a value Δ which is an upper bound on the maximum degree of the graph. In settings where Δ is not given, our algorithms may still be applied, using n in place of Δ; however, in this case, our energy and round complexity guarantees become worse. A more sophisticated approach to use when Δ is not known, is to guess a series of increasing values for Δ, running our algorithm for each guess. When the guesses are too small, portions of the output may fail to be independent, in which case affected vertices must detect this fact, and repeat the algorithm with the next, larger, value for Δ. The details are sufficiently complicated that we omit them from this version of the paper, mentioning only that using 2^{2^i} as the ith guess for Δ seems to work well, and carries an $O(\log \log n)$ factor overhead for our energy complexity, and an $O(1)$ factor overhead for our round complexity.

We assume all nodes start in the same state (with possibly no predesignated IDs) but have access to private random bits. (This allows nodes to locally generate $O(\log n)$-bit IDs which are unique with high probability.) Unless otherwise stated, we allow a failure probability of $1/\mathrm{poly}(n)$. We assume the standard RADIO-CONGEST model, which constrains the size of messages that can be sent in a single round, limiting them to $O(\log n)$ bits. This constraint reflects practical limitations in real-world networks, where bandwidth is limited, and large messages can cause congestion and delays. (The alternative model, where there is no such bandwidth constraint, is called the RADIO-LOCAL model.)

We assume synchronous communication where time is divided into discrete *rounds* (or timesteps). In each round, a processor can be in one of two states: *awake* or *sleeping*. In the awake state, a node can *either transmit* or *listen* (but not both due to the radio nature). In the sleeping model [13], only the awake rounds are counted towards the *energy complexity* (also called as *awake complexity* in some works – cf. Section 1.4). On the other hand, both sleeping and awake rounds are counted towards the *round complexity*. We assume local computation (performed by a node in a round) is free, but all our algorithms do local computation at a very small cost (at most logarithmic in n).

As in [18, 36], we assume *synchronous wake-up*, i.e., all nodes wake up simultaneously and can execute the algorithm immediately. (We note that *asynchronous wake-up* has also been studied in several prior works [33, 29, 34, 17]).

A message broadcast by a node u at time t is received by a neighbor v of u at round t if: (i) v listens at time t and (ii) no other neighbor of v transmits at time t. If some other neighbor of v also transmits at time t, v's reception will depend on how collisions are handled. Two standard and well-studied models of handling collisions are (i) with collision detection (CD) and (ii) without collision detection (no-CD) – defined as follows.

- **Collision-Detection (CD) model:** In the CD model, a listening node *can* distinguish between silence (no neighbors sending) and a collision (more than one neighbor sending). Thus, if more than one neighbor of v transmits, then v will hear a collision.
- **No Collision Detection (no-CD) model:** In the no-CD model a listening node *cannot* distinguish between silence and collisions between *two or more* messages. Thus, if more than one neighbor of v transmits, then v will hear nothing (i.e., silence).

[1] Although we informally think of n as being equal to the actual number of nodes in G, this is not at all necessary; the only disadvantage to having n much larger than the actual number of nodes is a possible increase in the time and energy complexities, and message lengths, of our algorithms.

The no-CD radio model is more challenging and has been used extensively in prior works (see e.g., [14, 4, 18, 34] and the references therein). We note that the CD model is also well-studied (see e.g., [4, 23, 36] and the references therein) and closely related to the well-studied *beeping model* [1], see Section 1.4. A no-CD algorithm will also work on the CD model, but may be less time- or energy-efficient than a respective CD algorithm.

1.2 Maximal Independent Set (MIS)

Maximal independent set (*MIS*) is one of the most well-studied problems in distributed graph algorithms. Given a graph with n nodes, each node must (irrevocably) commit to being in a subset $M \subseteq V$ (called the MIS) or not such that (i) every node is either in M or has a neighbor in M and (ii) no two nodes in M are adjacent to each other.

The MIS problem has been studied extensively for the last four decades in several distributed computing models (see e.g., [22, 20, 24, 25, 18] and the references therein). In this paper, our focus is on algorithms for the *radio network model* with low *energy complexity*.

1.3 Our Results

We present new, more *energy-efficient* MIS algorithms in radio networks on n nodes with *arbitrary* and unknown graph topology. Specifically, we obtain the following results:

1. **Lower Bound:** We show a lower bound of $\Omega(\log n)$ on the energy complexity. This lower bound applies to both the CD and no-CD models.
2. **Energy-Optimal MIS in the CD model**: We present a randomized distributed MIS algorithm with energy complexity $O(\log n)$, round complexity $O(\log^2 n)$, and failure probability $1/\mathrm{poly}(n)$. Our algorithm is energy-optimal because of the above lower bound.
3. **Energy-Efficient MIS in the no-CD model**: In the more restrictive no-CD model, we present a randomized distributed MIS algorithm with energy complexity $O(\log^2 n \log\log n)$, round complexity $O(\log^3 n \log \Delta)$, and failure probability $1/\mathrm{poly}(n)$. The energy complexity of our algorithm is significantly lower than the round (and energy) complexity of $O(\log^3 n)$ of the best known distributed MIS algorithm of Davies [18] for arbitrary graph topology. Furthermore, our energy complexity essentially matches (up to a $\log\log n$ factor) the best known lower bound for *round complexity* $\Omega(\log^2 n)$ [18, 21] in the no-CD model.

Our algorithms perform only unary communication, i.e., nodes only transmit a "1" bit if they transmit at all. In particular, our CD algorithm will also work in the simpler *beeping* model with the same energy and round complexities (cf. Section 3.1).

Our algorithms are an energy-efficient implementation of a Luby-like algorithm [31, 32] for radio networks. A somewhat straightforward implementation of Luby for radio networks will take $O(\log^2 n)$ energy and rounds in the CD model and $O(\log^4 n)$ energy and rounds in the no-CD model. In particular, in the more challenging no-CD model, it is non-trivial to improve the round complexity to $O(\log^3 n)$ rounds as was done in [18]; they do so by using an efficient radio implementation of Ghaffari's algorithm [22] for CONGEST (wired) networks. However, the algorithm of [18] also has $O(\log^3 n)$ energy complexity, as some nodes can be awake for so many rounds.

Our approach uses several non-trivial tools to implement a Luby-like algorithm energy-efficiently in a radio network – improving the above $O(\log^3 n)$ bound (in the no-CD model) by almost a logarithmic factor for energy complexity.

1.4 Additional Related Work

The literature on the MIS problem is vast. We focus mainly on those relevant to this work, i.e., *radio network model* and *energy-efficient* algorithms for *MIS*.

As mentioned earlier, almost all of the prior works focused on improving the *round complexity* of MIS in the radio model, and these focused on special families of graphs such as unit disk graphs [33] or bounded independence graphs [16, 17]. These algorithms typically assume the *no-CD* model and asynchronous wake-up and run in $O(\log^2 n)$ rounds. This bound can be improved for *multi-channel networks* [17], but the $O(\log^2 n)$ bound applies for standard single-channel networks (as assumed in this paper and many others cited above). It can be shown that $\Omega(\log^2 n)$ is a lower bound for the round complexity [18, 21]. For arbitrary graph topology, the best known complexity bound is $O(\log^3 n)$ [18]; this result, like ours, assumes synchronous wake-up. As pointed out in [18], the $\Omega(\log^2 n)$ round complexity lower bound applies to synchronous wake-up as well. In the CD model, the work of [36] showed a tight bound of $\Theta(\log n)$ on the *round complexity* of MIS in *growth-bounded* graphs. Note that our tight $\Theta(\log n)$ energy bound in the CD model applies to arbitrary graphs.

There also has been extensive work on the so-called *beeping* model [1, 28] where nodes can only communicate by beeping, which is equivalent to transmitting (or not transmitting) a bit (i.e., unary communication). Collision detection is assumed in the sense that if more than one neighbor of a node (say v) beeps, then v hears at least one beep. This is similar to the CD radio model, except in two ways: (i) In the radio model a node can send $O(\log n)$-sized message in a round, but a beep contains no information (except its presence). (ii) In the beeping model, the best known MIS algorithms typically assume *sender-side* collision detection, see e.g., the work of [28] which gives an optimal $O(\log n)$-round MIS algorithm in the beeping model. Sender-side collision detection means that the sender can detect a beep from its neighbors even when the sender is transmitting a beep. In the radio model, sender-side CD is not assumed – a node can only send or receive in any round, if they do both, then they will not hear anything.

The *sleeping* (or energy) model has been studied extensively in recent years. As mentioned in Section 1.1, the main feature of this model is that a node can be either in the sleeping or awake state in any round, and only rounds spent in the awake states are counted towards the energy (also called as *awake*) complexity. The energy model for *radio networks* (also called SLEEPING-RADIO [20]) used in this paper was introduced and studied in [12, 8] (inspired by earlier work on energy-efficient algorithms in radio networks e.g., [35, 29, 30]). Energy-efficient algorithms for radio networks for several problems such as broadcast, leader election, breadth-first search, maximal matching, diameter and minimum-cut computation have been studied [12, 8, 9, 14, 15, 11, 10, 7].

In another line of work, energy-efficient (or *awake-efficient*) algorithms for MIS have been designed in the *sleeping* model for CONGEST networks (called SLEEPING-CONGEST [20]). In this model, unlike radio networks (and more powerful), nodes can broadcast (or unicast) simultaneously without collisions; in other words, it is simply the standard CONGEST model with nodes having the additional flexibility to sleep to save energy. This model was introduced in [13]. This paper showed that MIS (in general graphs) can be solved in $O(1)$ *node-averaged* awake complexity, which is measured by the *average* number of rounds a node is awake. This is in contrast to the worst-case awake (or energy) complexity (that is used in this paper and all other papers cited here) which is the worst-case number of rounds a node is awake until it finishes the algorithm. The worst-case awake complexity of their MIS algorithm is $O(\log n)$, while the worst-case complexity (that includes all rounds, sleeping and awake) is $O(\log^{3.41} n)$ rounds. A question left open in [13] is whether one can

design an MIS algorithm with $o(\log n)$ worst-case awake complexity (even in the LOCAL model). This question was answered in [20] where it was shown that MIS can be solved in $O(\log \log n)$ awake complexity (with high probability) which is exponentially better than the round complexity lower bound of $\Omega(\sqrt{\log n / \log \log n})$. Several recent works in the SLEEPING-CONGEST model for fundamental problems such as MIS, approximate matching and vertex cover, spanning tree, minimum spanning tree, coloring, and other problems include [24, 6, 2, 25, 26, 19, 3].

1.5 Organization

The rest of the paper is organized as follows. We present our lower bound for the CD model in Section 2. We then present our CD algorithm in Section 3. Section 4 introduces preliminary techniques for our no-CD algorithm, which follows in Section 5. We note that in Sections 3–5, unless there is an explicit reference to an appendix, missing proofs can be found in the full version of the paper. We conclude and highlight open questions in Section 6.

2 An $\Omega(\log n)$ Energy Complexity Lower Bound for MIS

In this section, we show an $\Omega(\log n)$ lower bound on the energy complexity of MIS in the CD (and no-CD) model. Note that this lower bound is unconditional of the round complexity.

▶ **Theorem 1.** *In radio networks with CD, any algorithm that solves MIS with probability strictly more than $e^{-1/4}$ has energy complexity of at least $1/2 \log n$.*

Proof. Suppose to the contrary that every node is awake for $o(\log n)$ rounds and consider the following n-node anonymous graph G (assume that n is a multiple of 4): G is the union of $n/4$ disjoint edges and $n/2$ isolated nodes. Intuitively, in this graph, each isolated node must join the independent set whereas each non-isolated node must agree with its neighbor about which of them joins the independent set.

Consider a node v that is awake for only b rounds. If v hears nothing in those rounds, then by Bayes' Law, the conditional probability that v is an isolated node, given v's state of knowledge, is at least $1/2$. Therefore v must decide to join the independent set. On the other hand, let v be a non-isolated node and w be its neighbor. It is necessary for at least one message sent by either v or w to be heard by the other, since otherwise, both would join the independent set. We will show that, if $b = o(\log n)$, the probability that no messages are successfully heard by v or w is $n^{-o(1)}$. Since our graph contains $n/4$ disjoint pairs of this type, whose success or failure is independent, it follows that, with probability $1 - o(1)$, at least one such edge will fail to have either endpoint receive a message.

We define a strategy as a distribution over infinite sequences over the set $\{S, T, L\}$ with at most b occurrences of T and L combined, where S, T, and L correspond to sleep, transmit, and listen, respectively. A randomized algorithm (that uses at most b energy) corresponds to a strategy (or distribution) according to which each node will sample a sequence and follow it until it hears a message or a collision. All nodes are running the same algorithm, thus following the same strategy.

Let $A_{u,x}$ be the event that a node u's chosen random sequence (over $\{S, T, L\}$) agrees with an infinite sequence x over the set $\{T, L\}$ in each of its b occurrences of T or L, and let $I_{u,x}$ be the corresponding indicator random variable. Given any choice of string by u, if x is sampled uniformly at random, then $\mathbb{P}(A_{u,x}) \geq 2^{-b}$, since only the indices where u's string has T or L are relevant. Therefore $\mathbb{E}_x(I_{u,x}) \geq 2^{-b}$. Since this did not depend on the random

choices made by u, we have $\mathbb{E}_u(\mathbb{E}_x(I_{u,x})) \geq 2^{-b}$. Switching the order of the summation gives $\mathbb{E}_x(\mathbb{E}_u(I_{u,x})) \geq 2^{-b}$. By the probabilistic method, there exists an infinite sequence x^*, for which $\mathbb{E}_u(I_{u,x^*}) \geq 2^{-b}$; *i.e.*, $\mathbb{P}(A_{u,x^*}) \geq 2^{-b}$

Since nodes v and w sample their sequences (over $\{S, T, L\}$) independently and from the same distribution, $\mathbb{P}(A_{v,x^*}$ and $A_{w,x^*}) \geq 4^{-b}$.

The intersection of the events A_{v,x^*} and A_{w,x^*} implies that neither v nor w heard a message. During any timestep at which both were awake, their sequences agreed with the shared sequence x^*, so they both transmitted or listened. In this case, v and w would need to join the output set by the comments above. The probability that such an event occurs for at least one edge is given by

$$\begin{aligned}\mathbb{P}(\exists \text{ edge } (v,w) : \ v \text{ and } w \text{ both join}) &= 1 - \mathbb{P}(\forall \text{ edges } (v,w) : \ v \text{ and } w \text{ do not both join}) \\ &\geq 1 - \left(1 - 4^{-b}\right)^{n/4} \geq 1 - e^{-n/4^{b+1}}.\end{aligned}$$

This gives the algorithm at least a failure probability of at least $1 - e^{-1/4}$ if $b \leq 1/2 \log n$, so MIS requires $\Omega(\log n)$ energy. ◀

3 Energy-Optimal MIS in the Collision Detection (CD) Model

In the CD model, the best known algorithm solves MIS in $O(\log^2 n)$ round complexity, whereas the best known round complexity lower bound is $\Omega(\log n)$. The gap between both round complexities remains open, but in this section, we settle the energy complexity of MIS. More precisely, we show that MIS can be solved (energy-optimally) in $O(\log n)$ energy.

▶ **Theorem 2.** *In the CD model, Algorithm 1 outputs an MIS with probability at least* $1 - 1/n$. *Moreover, it does so using* $O(\log n)$ *energy and in* $O(\log^2 n)$ *rounds.*

Algorithm 1 MIS Algorithm in the CD model.

```
1:  status ← undecided
2:  for Luby phase i ← 1 to C log n do          ▷ C and β control the success probability
3:      x ← random string of β log n bits
4:      for Bitty phase j ← 1 to β log n do
5:          if x_j = 1 then
6:              transmit 1
7:          else
8:              listen
9:              if heard 1 or collision then
10:                 sleep for β log n − j rounds
11:                 break and jump to line 16
12:                                  ▷ Next line reached only if normal j loop termination
13:     transmit 1                               ▷ Confirm inclusion in the MIS
14:     status ← in-MIS
15:     Terminate
16:                                  ▷ Next line reached only if broke out of j loop
17:     listen                                   ▷ Final check in the current round
18:     if heard 1 or collision then
19:         status ← out-MIS
20:         Terminate
```

3.1 Algorithm Description

Our energy-optimal MIS algorithm (see Algorithm 1 and Figure 1 for a graphic illustration) runs in $C \log n$ *Luby phases* (Lines 3–20), each taking $\beta \log n + 1$ rounds (including sleeping and awake rounds) – where C and β are constants that control the success probability. Conceptually, we can separate each Luby phase into two parts: the *competition* part, consisting of the first $\beta \log n$ rounds, and the *checking* part, consisting of the last round. (The same conceptual structure applies to our later algorithms, see Section 5.)

The competition separates nodes into *winning* and *losing* nodes, and winners are added to the MIS in this Luby phase. Conditioned on the high probability that the ranks are distinct, the competition ensures that winners form an independent set among the non-terminated nodes. After all $\beta \log n$ rounds of the competition, the checking ensures that nodes in the computed independent set, and their neighbors, terminate respectively in and out of the MIS. More precisely, any node u that wins the competition enters the MIS, sends a message to all neighbors in the phase's last round, and terminates. Meanwhile, any node u that loses the competition listens in the last round and checks whether one of its neighbors won (i.e., if u hears a 1 or a collision). If that is the case, node u terminates as a node not in the MIS. Otherwise, it continues to the next Luby phase.

Next, we describe the competition. At the start, each non-terminated node u is awake and generates a sequence of $\beta \log n$ random bits independently. Call this $\beta \log n$ bit binary number the rank of u, denoted by x_u. Then, during the competition, node u determines whether its rank is smaller than the rank of any of its neighbors. This is done by a bit-by-bit comparison, using $\beta \log n$ *Bitty phases*. In the first such phase, node u examines the first bit of x_u, and if the bit is 1 then u transmits 1, otherwise u listens. In the latter case, if u hears a 1 or a collision, then u sleeps for all remaining Bitty phases (and has lost the competition). As for any subsequent phase i, any node u that hasn't gone to sleep executes the same procedure, but considering the ith bit of x_u. Finally, any node that has not gone to sleep in the $\beta \log n$ Bitty phases has won the competition.

Finally, we make two remarks. First, unlike the classical Luby's algorithm [31], in this version with the bit-by-bit competition, some winners may not be local maxima. However, the set of winners is a superset of the local maxima. Hence, the correctness of Luby's algorithm implies (see the proof of Lemma 5) that of this bit-by-bit version. Second, since in our algorithm, only the act of transmission matters, and the actual messages play no role, the algorithm can also be implemented in the beeping model with the same round and energy complexities. More concretely, in the pseudocode of Algorithm 1, one can replace "transmit 1" with "beep" and "heard 1 or a collision" with "heard a beep" since in the beeping model, a listening node hears a beep if at least one of its neighbors beep.

3.2 Analysis

We break our analysis of Algorithm 1 into two main steps. The proof of Theorem 2 appears in Appendix B. First, we prove that, with high probability, it finds an independent set.

▶ **Lemma 3.** *Let* $S = \{v\colon status(v) = \texttt{in-MIS}\}$ *be the set output by Algorithm 1. Then* $\mathbb{P}(S \text{ is an independent set}) \geq 1 - \frac{1}{2n}$.

Next, we show that, with high probability, every node is within one hop of at least one vertex output by our algorithm. To do so, we introduce the concept of a residual graph and follow part of a standard proof of Luby's algorithm, which says that, conditioned on whatever happened previously, one phase of Luby's algorithm shrinks the size of the residual graph by at least half, in expectation.

▶ **Definition 4.** *Let $V_0 = V$ and $1 \leq i \leq C \log n$. Let V_i denote the set of vertices that at the end of Luby phase i, have not yet terminated, or equivalently have status = `undecided`. We call the subgraph of G induced by V_i the* residual graph *at the end of Luby phase i, and denote it $G_i = (V_i, E_i)$.*

▶ **Lemma 5.** *Let $1 \leq i \leq C \log n$. Then, $\mathbb{E}(|E_i| \mid E_{i-1}) \leq \frac{|E_{i-1}|}{2}$.*

We remark that the proof of the above lemma (see Appendix B) does not imply that the running time of Luby's algorithm stochastically dominates the running time of Algorithm 1. This is because the monotonicity we exploit in our proof only exists within a single Luby phase; not across multiple Luby phases. However, we do get the following immediate corollary.

▶ **Corollary 6.** $\mathbb{P}(E_{C \log n} = \emptyset) \geq 1 - \frac{1}{2n}$.

▶ **Lemma 7.** *Let $S = \{v : status(v) = \texttt{in-MIS}\}$ be the set output by Algorithm 1. Then $\mathbb{P}(S \cup N(S) = V) \geq 1 - \frac{1}{2n}$.*

4 Auxiliary Primitives in the no-CD Model

We give several primitives in the no-CD model, and these will be key components of our no-CD MIS algorithm in Section 5. More concretely, we first give several energy-efficient backoff procedures. Then, we give an improved runtime version of the algorithm from [18].

4.1 Energy-Efficient Backoff Procedures

We design slightly more energy-efficient sender- and receiver-backoff procedures, which are used to adapt our CD algorithm to work in the no-CD model. These are described in more detail in Appendix C and have the following properties, which are proven in Appendix C.

▶ **Lemma 8.** *Let k be any positive integer. Both (sender-side and receiver-side) energy-efficient k-repeated backoffs take $O(k \log \Delta)$ rounds. Moreover, any node v calling*

- *Snd-EBackoff(k, Δ) is awake for k rounds.*
- *Rec-EBackoff$(k, \Delta, \Delta_{est})$ is awake for $O(k \log \Delta_{est})$ rounds.*

▶ **Lemma 9.** *For any node v calling Rec-EBackoff$(k, \Delta, \Delta_{est})$ with at most Δ_{est} neighbors calling Snd-EBackoff(k, Δ) simultaneously, it holds with probability at least $1 - (7/8)^k$ that node v returns true if and only if at least one of its neighbors simultaneously called Snd-EBackoff(k, Δ).*

4.2 Round-Efficient MIS in the no-CD Model

As stated in the related work, Davies [18] gives an $O(\log^3 n)$ round algorithm for MIS in the no-CD model. We make several (minor) modifications to improve its runtime to $O(\log^2 n \log \Delta)$ rounds, where Δ is an upper bound on the maximum degree of the graph. We call the improved algorithm LowDegreeMIS and give a short description below. It serves as one of the components of our $O(\log^2 n \log \log n)$ energy MIS algorithm given in Section 5.

Before we describe the minor improvements, note that in [18], each phase simulating one round of Ghaffari's MIS algorithm is also called a "round," whereas timesteps are what we call rounds in our model. First, the number of timesteps in the Decay subroutine can be reduced to $\Theta(\log \Delta)$. Even then, $O(\log n)$ iterations of this shorter Decay subroutine results in the same high probability success guarantee. Second, the EstimateEffectiveDegree

subroutine can be run for only $\Theta(\log \Delta)$ outer loop iterations (each still consisting of $O(\log n)$ timesteps) while maintaining the same high probability success guarantee. The rest of the analysis remains unchanged.

5 Energy-Efficient MIS in the no-CD Model

In the no-CD model, the best known round and energy upper bound is $O(\log^3 n)$ [18], or if we parametrize by Δ, $O(\log^2 n \log \Delta)$ (see Section 4.2). On the other hand, the only known energy lower bound in this setting is the $\Omega(\log n)$ lower bound from the CD model. Hence, there is a $\Theta(\log n \log \Delta)$ gap between the known upper and lower bounds in the no-CD model. In this section, we present an MIS algorithm for no-CD (Algorithm 2) with a significantly better energy complexity of $O(\log^2 n \log \log n)$, and as a result we narrow this gap by a logarithmic factor, down to $\Theta(\log n \log \log n)$.

▶ **Theorem 10.** *In the no-CD model, Algorithm 2 outputs an MIS with probability at least* $1 - 1/n$. *Moreover, it does so using* $O(\log^2(n) \log \log n)$ *energy and in* $O(\log^3 n \log \Delta)$ *rounds.*

5.1 Insights into our Algorithm

To solve MIS in the no-CD setting, one option is to take an MIS algorithm in the CD model (say, Algorithm 1) and simulate it using traditional backoff (described in Section 4.1). However, we would need to simulate each round with high probability, leading to an $O(\log n \log \Delta)$ blow-up in the round complexity but also, and most importantly, in the energy complexity. We identify two areas that drive the energy cost up by a factor of $O(\log n \log \Delta)$ when we take the above approach with Algorithm 1 and make non-trivial adaptations to address them.

5.1.1 Competition

The first problematic place is in the inner loop of the competition, where nodes with a '0' bit listen to determine if they should drop out. An eventual winner is one that survives the Luby phase without hearing any of its neighbors. Note that just as any other node, it is likely to have $\Theta(\log n)$ 0's in its bitstring (since each bit is chosen uniformly at random and independently). But, because an eventual winner never hears a neighbor, it will listen for all rounds in all backoffs corresponding to '0' bits. As a result, during this Luby phase, it will have spent $O(\log^2 n \log \Delta)$ energy.

A similar issue may affect some eventual losers, as well. For example, two adjacent nodes may happen to choose random strings that agree in their first $\Theta(\log n)$ bits. If they have no other neighbors to knock them out of competition, both nodes will spend $\Theta(\log^2 n \log \Delta)$ energy listening to silence during their 0 bit rounds, before one eventually loses.

To fix the above issues, we give each node an energy budget of $O(\log n \log \Delta)$. If a node is awake for the entire backoff corresponding to its first '0' bit, then it has used up a significant portion of its budget for the entire algorithm. It cannot afford to use this much energy repeatedly, so it will do two things to reduce its energy use: it will play the remainder of the Luby phase with the (justified) assumption that it has only $O(\log n)$ surviving neighbors, so it can shorten how long it listens in the backoffs. Additionally, even if it is knocked out of the competition, it will commit itself to terminating in this Luby phase. The nodes that are thus committed induce a subgraph of maximum degree $O(\log n)$, so at the end of the Luby phase, they can afford to run an algorithm that is energy-efficient on small degree graphs (e.g,. the naive simulation of Algorithm 1, or for better runtime, the algorithm from Section 4.2). Each committed node with no neighbor in the output set runs this subroutine exactly once, bringing the energy complexity to $O(\log^2 n \log \log n)$.

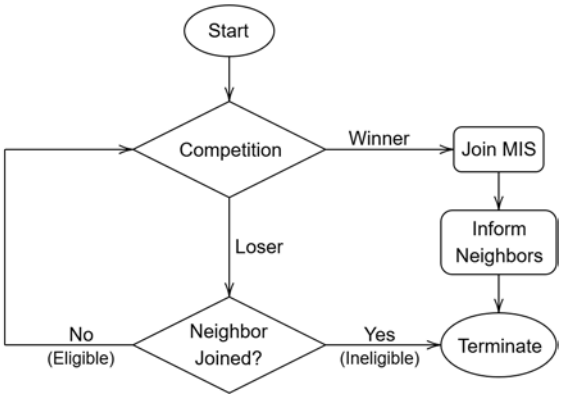


Figure 1 Flowchart for our CD algorithm, Algorithm 1.

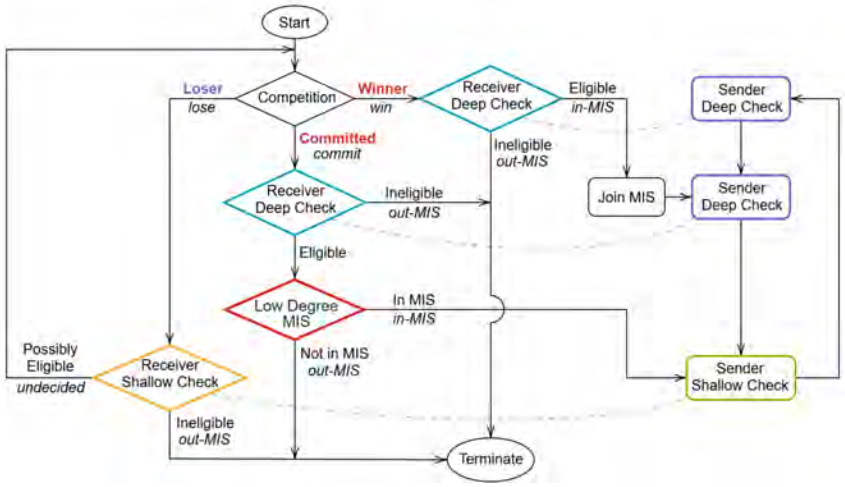


Figure 2 Flowchart for our no-CD algorithm, Algorithm 2. Dashed lines indicate concurrent execution. Energy usage is color-coded as follows: $O(\log^2 n \log\log n)$, $O(\log n \log \Delta)$, $O(\log n)$, $O(\log \Delta)$, $O(1)$.

5.1.2 Checking

The second area of concern is in the notification process at the end of each Luby phase. All nodes who lost in the competition now listen, because they might have a neighbor in the output set. However, if they have no such neighbors, then they listen for the full $O(\log n \log \Delta)$ rounds. Crucially, if this happens in every phase, nodes would end up using too much energy. And yet, this seems necessary, as we should ensure that any node that neighbors an MIS node no longer participates in the following competitions.

However, we give up on that property. More concretely, at the end of each Luby phase, losing nodes perform a *"shallow" check* to detect the presence of MIS neighbors with only constant probability – via a single iteration of backoffs. This gives the neighbors of MIS nodes a constant probability to drop out, but at a vastly reduced cost. On the other hand, a winning node performs a thorough, *"deep" check* – via $O(\log n)$ iterations of backoffs – to detect the presence of MIS neighbors with high probability – to decide whether it joins the output set (when there are no such neighbors) or sets its status to "not in MIS."

5.2 Algorithm

Next, we give our MIS algorithm: Algorithm 2. Figure 2 provides a graphic illustration of the flow control for Algorithm 2. This algorithm relies on several constants. Some ensure our algorithm successfully computes an MIS with probability at least $1 - 1/n$. Increasing these constants yields better success probabilities. In particular, we choose $\beta \geq 4$, $\kappa \geq 5$ and $C \geq 4/\log(64/63)$, whereas we choose C' such that Rec-EBackoff$(C' \log n, \Delta)$ succeeds with probability $1 - 1/n^5$. As for LowDegreeMIS, we ensure it succeeds with probability $1 - 1/n^2$.

Our other constants ensure that nodes stay synchronized throughout the Luby phase, and upper bound the round complexity for

- Snd-EBackoff(k, Δ) and Rec-EBackoff(k, Δ): $T_B(k) = k\lceil \log \Delta \rceil$,
- Competition(Δ): $T_C = \beta \log^2 n \log \Delta$,
- LowDegreeMIS on an induced subgraph of max degree $\kappa \log n$: $T_G = O(\log^2 n \log\log n)$,
- A single Luby phase: $T_L = T_C + 2T_B(C' \log n) + T_G + T_B(1) = O(\log^2 n \log \Delta)$.

Algorithm 2 Distributed Maximal Independent Set (MIS) no-CD Algorithm.

```
1:  status ← undecided
2:  for Luby phase i ← 1 to C log n do
3:      if status = undecided then Competition(Δ)
4:      else sleep until round (i − 1)T_L + T_C

5:      heard ← False                                   ▷ All nodes are synchronized here
6:      if status = in-MIS then Snd-EBackoff(C′ log n, Δ)
7:      else if status = win then
8:          heard ← Rec-EBackoff(C′ log n, Δ)           ▷ Deep check for MIS neighbors
9:          if heard then status ← out-MIS, then terminate early
10:         else status ← in-MIS
11:     else sleep until round (i − 1)T_L + T_C + T_B(C′ log n)

12:     if status = in-MIS then                         ▷ All nodes are synchronized here
13:         Snd-EBackoff(C′ log n, Δ)
14:         sleep until round (i − 1)T_L + T_C + 2T_B(C′ log n) + T_G
15:     else if status = commit then
16:         heard ← Rec-EBackoff(C′ log n, Δ)           ▷ Deep check for MIS neighbors
17:         if heard then
18:             status ← out-MIS, then terminate early
19:         else
20:             status ← undecided
21:             LowDegreeMIS          ▷ Run on subgraph of maximum degree O(log n)
22:     else sleep until round (i − 1)T_L + T_C + 2T_B(C′ log n) + T_G

23:     if status = in-MIS then Snd-EBackoff(1, Δ)      ▷ All nodes are synchronized here
24:     else
25:         heard ← Rec-EBackoff(1, Δ)                  ▷ Shallow check for MIS neighbors
26:         if heard then status ← out-MIS, then terminate early
27:         else status ← undecided
```

Algorithm 3 COMPETITION.

```
1: procedure Competition(Δ)
2:     Δ_est ← Δ, heard ← False
3:     x ← random string of β log n bits
4:     for Bitty phase j ← 1 to β log n do
5:         if status = lose then sleep
6:         else if x_j = 1 then Snd-EBackoff(C' log n, Δ)
7:         else
8:             heard ← heard ∨ Rec-EBackoff(C' log n, Δ, Δ_est)        ▷ Logical OR
9:             if heard and status ≠ commit then
10:                status ← lose
11:            else if not heard then  ▷ Not hearing implies O(log n) undecided neighbors.
12:                Δ_est ← min{Δ, κ log n}
13:                status ← commit
14:    if not heard then        ▷ Nodes that heard nothing win, including committed ones.
15:        status ← win
```

5.3 Properties of a Luby Phase

We now show properties that pertain to a single Luby phase i. More precisely, we show properties on sets C_i and W_i – these are defined as the set of undecided nodes that run the competition in the ith Luby phase and subsequently set their status to `commit` and `win`, respectively. First, we consider the sets C_i and show that neighboring, committed nodes must have committed in the same Bitty phase with high probability. This helps us show that for any Luby phase i, C_i induces a logarithmic degree subgraph. That is, with high probability, among the neighbors of any given committed node v, there can be at most $O(\log n)$ nodes with *status* $\neq$ `lose`. This justifies our reduction of the degree estimate to $\kappa \log n$ (in our case, $\kappa \geq 5$) whenever a node sets its status to `commit`.

▶ **Lemma 11.** *Consider a single call to* Competition. *Let u, v be two neighboring nodes that set their status to* `commit`. *Then, with probability at least $1 - 2/n^5$, u and v set their status to* `commit` *in the same Bitty phase.*

▶ **Lemma 12.** *Consider a single call to* Competition. *Let κ be any strictly positive integer, and let B be the event that there exists a node v such that more than $\kappa \log n$ neighbors of v do not have status* `lose` *in the (Bitty) phase of v's first 0 bit, and v sets its status to* `commit`. *Then, $\mathbb{P}(B) \leq 4/n^4$.*

▶ **Corollary 13.** *For any Luby phase i, each statement below holds with probability $1 - 4/n^4$:*

1. *During the competition, for Bitty phase j and node v (such that v's status is not* `lose`*), the degree estimate of v upper bounds the number of v's awake neighbors (i.e., starting the Bitty phase with status $\neq$* `lose`*).*
2. *The subgraph induced by C_i has maximum degree $O(\log n)$.*

Let C_i^* denote those nodes with status `commit` but having not detected any MIS neighbor during Luby phase i. Then, Corollary 13 implies that we can run the $O(\log^2 n \log \Delta)$ round MIS algorithm described in Section 4.2, to compute an MIS on the subgraph (of the communication graph) induced by C_i^* in $O(\log^2 n \log \log n)$ rounds (and energy).

We follow up by showing properties on the set W_i for any Luby phase i. We show that in the competition, the undecided nodes with locally maximum bitstrings set their status to `win` (and thus join set W_i) with high probability. This implies, among other things, that W_i is not empty until no undecided nodes remain. Next, we show that in the competition, with high probability, no two neighbors set their status to `win` (i.e., W_i is independent).

▶ **Lemma 14.** *Consider a single call to* Competition*. Let v be an undecided node whose bitstring $x(v)$ is a local maximum. Then, $\mathbb{P}(v \text{ sets status to } \texttt{win}) \geq 1 - 1/n^2$.*

▶ **Lemma 15.** *For any Luby phase i and two neighbors u, v: $\mathbb{P}(u \in W_i \text{ and } v \in W_i) \leq 6/n^4$.*

Finally, we highlight that any node that attempts to join the MIS following the competition (i.e., that is in $W_i \cup C_i$) is decided by the end of that Luby phase.

▶ **Lemma 16.** *For any Luby phase i, any node in $W_i \cup C_i$ decides by the end of that phase with probability 1.*

5.4 Analysis

We now prove our main result, Theorem 10. We start with auxiliary lemmas that help prove the correctness. Their proofs can be found in Appendix B. First, we show that the set of nodes with status `in-MIS` stays independent throughout the algorithm with high probability.

▶ **Lemma 17.** *For any Luby phase i, at the start, the nodes with status `in-MIS` form an independent set with probability at least $1 - O(\log n)/n^2$.*

It remains to show that the logarithmic number of Luby phases of Algorithm 2 suffices for all nodes to become decided with high probability (as either `in-MIS` or `out-MIS`). To do so, we follow the lines of the classical Luby analysis. In other words, we consider the residual graphs, whose definition follows, and show that the number of edges in the residual graphs decreases by a constant fraction every phase (see Lemma 20).

▶ **Definition 18.** *Let $V_0 = V$ and $1 \leq i \leq C \log n$. Let V_i denote the set of vertices that at the end of Luby phase i, have status $\neq$ `out-MIS`. We call the subgraph of G induced by V_i the* residual graph *at the end of Luby phase i, and denote it $G_i = (V_i, E_i)$.*

Note that, in contrast to the definition in Section 3, the residual graph contains the decided MIS nodes – as here, MIS nodes do not terminate early – as well as undecided nodes (i.e., with *status* = `undecided`) that have an MIS neighbor but do not know it yet. (This can happen because MIS nodes inform their neighbors via shallow checks, which only succeed with constant probability per phase.) In particular, the latter nodes complicate the analysis: they continue to participate in the competition but cannot enter the MIS, and yet, the following lemma shows that such nodes have a limited impact on the progress of the algorithm. In short, they lead to a constant factor slowdown only.

▶ **Lemma 19.** *For any Luby phase $1 \leq i \leq C \log n$, $\mathbb{E}(X_i \mid E_{i-1}) \geq \frac{|E_{i-1}|}{8}$.*

After showing the above lemma, we can prove that every phase leads to, in expectation, a constant factor loss in the (edge) size of the residual graphs.

▶ **Lemma 20.** *For any Luby phase $1 \leq i \leq C \log n$, $\mathbb{E}(|E_i| \mid E_{i-1}) \leq \frac{63}{64}|E_{i-1}|$.*

Finally, we prove our main result.

▶ **Theorem 10.** *In the no-CD model, Algorithm 2 outputs an MIS with probability at least* $1-1/n$*. Moreover, it does so using* $O(\log^2(n)\log\log n)$ *energy and in* $O(\log^3 n \log\Delta)$ *rounds.*

Proof. First, we show correctness. By Lemma 17, with probability at least $1-O(\log n)/n^2 \geq 1-1/4n$, the set of nodes with status `in-MIS` is independent throughout the execution of Algorithm 2. Hence, it suffices to show that all nodes become decided within $C\log n$ Luby phases. By induction and Lemma 20, for every $i \geq 1$, we have $\mathbb{E}(|E_i|) \leq \left(\frac{63}{64}\right)^i |E_0|$. Hence, by Markov's inequality, $\mathbb{P}(|E_i| \geq 1) \leq \mathbb{E}(|E_i|) \leq \left(\frac{63}{64}\right)^i |E_0|$. By choosing $C \geq 4/\log(64/63)$, we get that $\mathbb{P}(E_{C\log n} = \emptyset) \geq 1-1/4n$. Finally, we consider any Luby phase i with $E_i = \emptyset$. In that phase's competition, any undecided node chooses a local maximum bitstring and enters W_i with probability at least $1-1/4n$ by Lemma 14, in which case it becomes decided by the end of the phase with probability 1, by Lemma 16. (A final $1/4n$ probability term comes from thresholding the energy complexity, as explained at the end of the proof.) In summary, all nodes become decided, and the output is an MIS with probability at least $1-1/n$.

Second, we bound the round complexity. From the algorithm description, each Luby phase takes $T_L = O(\log^2(n)\log\Delta)$ rounds. Hence, the round complexity of the algorithm (which runs for $C\log n$ Luby phases) is $O(\log^3 n \log\Delta)$ rounds.

Finally, we upper bound the energy complexity. First, if any node v starts the Luby phase as an MIS node then v spends $O(\log n)$ energy: v sleeps during the competition, and its participation in sender backoffs during the checking adds up to $O(\log n)$ energy by Lemma 8.

Second, if node v starts `undecided` and enters $C_i \cup W_i$, then during that Luby phase, v spends $O(\log^2(n)\log\log n)$ energy. Indeed, during the competition, node v uses $O(\log^2 n)$ energy for the sender backoffs overall (by Lemma 8), and $O(\log^2 n) + O(\beta\log n \cdot \log(n)\log\log n)$ energy for the receiver backoffs overall (by Lemma 8 and due to the change in the degree estimate). As for the checking, node v uses up $O(\log^2(n)\log\log n)$ energy during LowDegreeMIS due to Davies' algorithm (cf. Section 4.2) and because the subgraph induced by C_i has maximum degree $O(\log n)$ (see Corollary 13).

By Lemma 16, there can be a single Luby phase in which v starts `undecided` and enters $C_i \cup W_i$, as subsequently node v either sets its status to `in-MIS`, or sets its status to `out-MIS` and sleeps for the remainder of the MIS algorithm. Hence, summing over all Luby phases in which v either starts as an MIS node, or attempts to join the MIS (i.e., joins $C_i \cup W_i$), the energy spent by v is upper bounded by $O(\log^2(n)\log\log n)$.

It remains to bound the energy complexity spent over any Luby phases in which v starts undecided and loses the competition. Let A_i be the energy spent by v in Luby phase i times the indicator random variable that v loses Luby phase i. We can upper bound the energy complexity $A = \sum_{i=1}^{\beta\log n} A_i$ by $O(\log^2 n)$ with high probability. Note that the energy spent in Luby phases in which v loses is either spent sending during the leading 1 bits or listening during the first 0 bit. Let X_i be the number of leading 1 bits in v's bitstring for Luby phase i. The random variables (X_i+1) are independent geometric random variables with parameter $\frac{1}{2}$. In the Bitty phase corresponding to v's first 0 bit in Luby phase i, let N_i be the number of v's neighbors that are sending because they are still active and have a 1 bit and let B_i be the number of backoffs v must participate in until hearing a message. Let I_i be the indicator random variable that $N_i > 0$ and let $Y_i = B_i I_i + (1-I_i)$. Conditioned on N_i and any random variables Y_j for $j < i$, Y_i is a geometric random variable with parameter $p(N_i)$, where $p(0) = 1$ and $p(n) \geq \frac{1}{8}$ for $n \geq 1$, which follows from well-known statements on exponential backoff (see Lemma 9 in Section 4.1). Then, $A_i \leq O(\log n)X_i + O(\log\Delta)Y_i$.

For the X_i random variables, we instead bound the sum of (X_i+1) random variables, since they are more nicely defined. Let $X = \sum_{i=1}^{\beta\log n}(X_i+1)$. Then, $\mu_X = \mathbb{E}[X] = 2\beta\log n$. By Theorem 23, we have $\mathbb{P}(X \geq 2\beta\lambda\log n) \leq e^{-\beta\log n(\lambda-1-\ln\lambda)} \leq n^{-C_X}$ for arbitrary C_X and

sufficiently large λ. Next, we want to bound the sum of the Y_i random variables. However, these Y_i random variables are not independent since the number of backoffs required to hear a message depends on the number of neighbors sending, and the number of neighbors sending in any Luby phase affects how many neighbors send in the next Luby phase. Hence, we instead show by induction that $Y = \sum_{i=1}^{\beta \log n} Y_i$ is stochastically dominated by $Z = \sum_{i=1}^{\beta \log n} Z_i$, where the random variables Z_i are independent geometric random variables with parameter $\frac{1}{8}$. For the base case, it can be easily seen that $Y_1 \preceq Z_1$. Suppose $\sum_{i=1}^{k-1} Y_i \preceq \sum_{i=1}^{k-1} Z_i$. Then,

$$\begin{aligned}
\mathbb{P}\left(\sum_{i=1}^{k} Y_i \geq y\right) &= \sum_{y',n} \mathbb{P}\left(\sum_{i=1}^{k} Y_i \geq y \,\middle|\, N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \mathbb{P}\left(N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \\
&= \sum_{y',n} \mathbb{P}\left(Y_k \geq y - y' \,\middle|\, N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \mathbb{P}\left(N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \\
&= \sum_{y',n} \min\left\{1, (1-p(n))^{y-y'-1}\right\} \mathbb{P}\left(N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \\
&\leq \sum_{y',n} \min\left\{1, \left(\frac{7}{8}\right)^{y-y'-1}\right\} \mathbb{P}\left(N_i = n, \sum_{i=1}^{k-1} Y_i = y'\right) \\
&= \mathbb{P}\left(Z_k + \sum_{i=1}^{k-1} Y_i \geq y\right) \leq \mathbb{P}\left(\sum_{i=1}^{k} Z_i \geq y\right),
\end{aligned}$$

where the first inequality comes from $p(n) \geq \frac{1}{8}$ and the second comes from Lemma 22. Applying Theorem 23 to Z with $\mu_Z = \mathbb{E}[Z] = 8\beta \log n$, we get $\mathbb{P}(Z \geq 8\beta\lambda \log n) \leq e^{-\beta \log n(\lambda - 1 - \ln \lambda)} \leq n^{-C_Z}$ for arbitrary C_Z and sufficiently large λ. Following which, the stochastic domination implies the same bound holds for $\mathbb{P}(Y \geq 8\beta\lambda \log n)$.

Finally, a union bound (over the nodes) shows that $A = O(\log^2 n)$ with high probability, say $1 - 1/4n$. Adding up the energy complexities, we get that all nodes spend $O(\log^2(n) \log\log n)$ energy with probability at least $1 - 1/4n$ during this MIS algorithm. To obtain the claimed deterministic upper bound on the energy complexity, we can do the following: any node spending more than some $\Theta(\log^2(n) \log\log n)$ energy threshold simply sleeps for the remainder of the algorithm and arbitrarily decides whether to join the MIS. ◀

6 Conclusion and Open Questions

We presented new, more energy-efficient MIS algorithms for radio networks. While our CD algorithm is energy-optimal, it is unclear whether our no-CD algorithm is. A key open problem is whether we can improve the energy complexity significantly in the no-CD model or whether our bound of $O(\log^2 n \log\log n)$ is nearly optimal (up to a $O(\log\log n)$ factor).

In the CD model, it is known that one can design an $O(\log n)$ round algorithm for growth-bounded graphs [36]. It is not clear if this bound can be achieved for general graphs as well. The round complexity of our CD algorithm is $O(\log^2 n)$, which is off by an $O(\log n)$ factor of the $\Omega(\log n)$ round complexity lower bound in the CD model [36]. Can we improve the round complexity in the CD model while getting the optimal $O(\log n)$ energy bound?

In the no-CD model, our energy complexity bound of $O(\log^2 n \log\log n)$ almost matches the round complexity lower bound of $\Omega(\log^2 n)$. Can we design a better energy-efficient algorithm that takes $o(\log^2 n)$ energy in arbitrary graphs? Furthermore, can we improve the round complexity of our no-CD algorithm while maintaining its energy complexity?

References

1 Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Computing*, 26(4):195–208, 2013. doi:10.1007/S00446-012-0175-7.

2 John Augustine, William K. Moses Jr., and Gopal Pandurangan. Awake complexity of distributed minimum spanning tree. In Yuval Emek, editor, *Structural Information and Communication Complexity - 31st International Colloquium, SIROCCO 2024, Vietri sul Mare, Italy, May 27-29, 2024, Proceedings*, volume 14662 of *Lecture Notes in Computer Science*, pages 45–63. Springer, 2024. doi:10.1007/978-3-031-60603-8_3.

3 Alkida Balliu, Pierre Fraigniaud, Dennis Olivetti, and Mikaël Rabie. Solving sequential greedy problems distributedly with sub-logarithmic energy cost. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2025.

4 Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. In Jean-Claude Bermond and Michel Raynal, editors, *Distributed Algorithms*, pages 24–32, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:10.1007/3-540-51687-5_29.

5 Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992. doi:10.1016/0022-0000(92)90042-H.

6 Leonid Barenboim and Tzalik Maimon. Deterministic logarithmic completeness in the distributed sleeping model. In *35th International Symposium on Distributed Computing, DISC*, volume 209, pages 10:1–10:19, 2021. doi:10.4230/LIPICS.DISC.2021.10.

7 Yi-Jun Chang. The energy complexity of diameter and minimum cut computation in bounded-genus networks. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity (SIROCCO)*, pages 262–296, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-32733-9_12.

8 Yi-Jun Chang, Varsha Dani, Thomas P. Hayes, Qizheng He, Wenzheng Li, and Seth Pettie. The energy complexity of broadcast. In *ACM PODC*, pages 95–104, 2018. doi:10.1145/3212734.3212774.

9 Yi-Jun Chang, Varsha Dani, Thomas P. Hayes, and Seth Pettie. The energy complexity of BFS in radio networks. In *ACM PODC*, pages 273–282, 2020. doi:10.1145/3382734.3405713.

10 Yi-Jun Chang, Ran Duan, and Shunhua Jiang. Near-optimal time–energy tradeoffs for deterministic leader election. *ACM Trans. Algorithms*, 19(4), September 2023. doi:10.1145/3614429.

11 Yi-Jun Chang and Shunhua Jiang. The energy complexity of las vegas leader election. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, pages 75–86, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490148.3538586.

12 Yi-Jun Chang, Tsvi Kopelowitz, Seth Pettie, Ruosong Wang, and Wei Zhan. Exponential separations in the energy complexity of leader election. *ACM Trans. Algorithms*, 15(4):49:1–49:31, 2019. Conference version: *ACM STOC* 2017. doi:10.1145/3341111.

13 Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. Sleeping is efficient: MIS in $O(1)$-rounds node-averaged awake complexity. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 99–108, 2020. doi:10.1145/3382734.3405718.

14 Varsha Dani, Aayush Gupta, Thomas P. Hayes, and Seth Pettie. Wake up and Join Me! an Energy-Efficient Algorithm for Maximal Matching in Radio Networks. In *35th International Symposium on Distributed Computing (DISC)*, pages 19:1–19:14, 2021. doi:10.4230/LIPICS.DISC.2021.19.

15 Varsha Dani and Thomas P. Hayes. How to wake up your neighbors: Safe and nearly optimal generic energy conservation in radio networks. In Christian Scheideler, editor, *36th International*

Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA, volume 246 of *LIPIcs*, pages 16:1–16:22, 2022. doi:10.4230/LIPICS.DISC.2022.16.

16 Sebastian Daum, Mohsen Ghaffari, Seth Gilbert, Fabian Kuhn, and Calvin Newport. Maximal independent sets in multichannel radio networks. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 335–344, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2484239.2484257.

17 Sebastian Daum and Fabian Kuhn. Tight bounds for mis in multichannel radio networks. In Yoram Moses, editor, *Distributed Computing*, pages 573–587, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-48653-5_38.

18 Peter Davies. Uniting general-graph and geometric-based radio networks via independence number parametrization. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, PODC '23, pages 290–299, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3583668.3594595.

19 Fabien Dufoulon, Pierre Fraigniaud, Mikaël Rabie, and Hening Zheng. Distributed coloring in the sleeping model, 2024. doi:10.48550/arXiv.2405.10058.

20 Fabien Dufoulon, William K. Moses Jr., and Gopal Pandurangan. Distributed MIS in O(log log n) awake complexity. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 135–145. ACM, 2023. doi:10.1145/3583668.3594574.

21 Martin Farach-Colton, Rohan J. Fernandes, and Miguel A. Mosteiro. Lower bounds for clear transmissions in radio networks. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 447–454. Springer, 2006. doi:10.1007/11682462_42.

22 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 270–277, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974331.CH20.

23 Mohsen Ghaffari, Bernhard Haeupler, and Majid Khabbazian. Randomized broadcast in radio networks with collision detection. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 325–334, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2484239.2484248.

24 Mohsen Ghaffari and Julian Portmann. Average awake complexity of MIS and matching. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 45–55, 2022. doi:10.1145/3490148.3538566.

25 Mohsen Ghaffari and Julian Portmann. Distributed MIS with low energy and time complexities. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 146–156. ACM, 2023. doi:10.1145/3583668.3594587.

26 Mohsen Ghaffari and Anton Trygub. A near-optimal low-energy deterministic distributed sssp with ramifications on congestion and apsp. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 401–411, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3662158.3662812.

27 Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135:1–6, 2018. doi:10.1016/j.spl.2017.11.017.

28 Peter Jeavons, Alex Scott, and Lei Xu. Feedback from nature: simple randomised distributed algorithms for maximal independent set selection and greedy colouring. *Distributed Computing*, 29(5):377–393, 2016. doi:10.1007/S00446-016-0269-8.

29 Tomasz Jurdzinski, Miroslaw Kutylowski, and Jan Zatopianski. Efficient algorithms for leader election in radio networks. In Aleta Ricciardi, editor, *Proceedings of the Twenty-First Annual*

ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002, pages 51–57. ACM, 2002. doi:10.1145/571825.571833.

30 Marcin Kardas, Marek Klonowski, and Dominik Pajak. Energy-efficient leader election protocols for single-hop radio networks. In *42nd International Conference on Parallel Processing, ICPP 2013, Lyon, France, October 1-4, 2013*, pages 399–408. IEEE Computer Society, 2013. doi:10.1109/ICPP.2013.49.

31 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

32 Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, and Akka Zemmari. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing*, 23(5-6):331–340, 2011. doi:10.1007/S00446-010-0121-5.

33 Thomas Moscibroda and Roger Wattenhofer. Efficient computation of maximal independent sets in unstructured multi-hop radio networks. In *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE Cat. No.04EX975)*, pages 51–59, 2004. doi:10.1109/MAHSS.2004.1392071.

34 Thomas Moscibroda and Roger Wattenhofer. Maximal independent sets in radio networks. In Marcos Kawazoe Aguilera and James Aspnes, editors, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, PODC 2005, Las Vegas, NV, USA, July 17-20, 2005*, pages 148–157. ACM, 2005. doi:10.1145/1073814.1073842.

35 Koji Nakano and Stephan Olariu. Randomized leader election protocols in radio networks with no collision detection. In D. T. Lee and Shang-Hua Teng, editors, *Algorithms and Computation, 11th International Conference, ISAAC 2000, Taipei, Taiwan, December 18-20, 2000, Proceedings*, volume 1969 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 2000. doi:10.1007/3-540-40996-3_31.

36 Johannes Schneider and Roger Wattenhofer. What is the use of collision detection (in wireless networks)? In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing*, pages 133–147, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-15763-9_14.

37 Dorothea Wagner and Roger Wattenhofer, editors. *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*. Springer, 2007.

A Some Useful Facts from Probability Theory

We will need the following concepts and results from Probability Theory.

▶ **Definition 21 (Stochastic Domination).** *For any two random variables X, Y, we say that X stochastically dominates Y, denoted by $X \succeq Y$, if for all $z \in \mathbb{R}$, $\mathbb{P}[X > z] \geq \mathbb{P}[Y > z]$.*

In general it is not true that stochastic domination carries over to sums of random variables. However, the following lemma shows that it is true in a limited setting. We leave its proof as an exercise to the reader.

▶ **Lemma 22.** *Let X and Y be random variables such that $X \succeq Y$ and let Z be a random variable that is independent of X and Y. Then, $X + Z \succeq Y + Z$.*

We will also need the following concentration inequality for the sum of independent geometric random variables.

▶ **Theorem 23** (Theorem 2.1 from [27]). *Let $X_1, X_2, \ldots, X_n$ be independent geometric random variables with parameters $p_1, p_2, \ldots, p_n$ respectively. Let $X = \sum_{i=1}^{n} X_i$, $\mu = E[X]$, and $p_* = \min_i p_i$. Then, for any $\lambda \geq 1$ the following holds:*

$$\mathbb{P}\left(X \geq \lambda\mu\right) \leq e^{-p_*\mu(\lambda - 1 - \ln \lambda)}$$

B Proofs

Proof of Lemma 5. First, observe that, in each Luby phase, Algorithm 1 always adds all local maxima of the random function x to the output. This is because, for a local maximum v, every neighbor has a zero in the first Bitty phase when it disagrees with v. Consequently, if you started both Algorithm 1 and Luby's algorithm from the same residual graph G_{i-1}, Algorithm 1 would remove a superset of the vertices removed by Luby's algorithm. Since we are looking at vertex-induced subgraphs, this implies we also remove a superset of the edges removed by Luby's algorithm. Since Luby's algorithm already satisfies the conclusion of the Lemma, so does Algorithm 1. ◀

Proof of Theorem 2. The correctness and round complexity claims are straightforward. For the correctness claim, by Lemma 3, with high probability, the output is an independent set. When it is, Lemma 7 tells us that, with high probability, it is also maximal. Thus, a union bound over the failure probabilities leads to the claimed correctness guarantee. As for the round complexity, Algorithm 1 consists of two nested for-loops, each with $O(\log n)$ iterations, which implies the claimed $O(\log^2 n)$ bound.

Finally, we show the energy complexity upper bound. More concretely, we show that, with probability $1 - 1/\operatorname{poly}(n)$, no node spends more than $O(\log n)$ energy. Since our algorithm already has a $1/\operatorname{poly}(n)$ chance of failure, we can think of exceeding the energy bound as another way to fail, in which case our bound on the maximum energy spent is absolute.

Let us examine the energy spent by a particular node, v. We split our analysis into two parts: early rounds, in which v's decision is still in doubt, and late rounds, in which v's decision is determined, although v may not know it yet. Specifically, a *late round* is a Bitty phase for which v is still active, but all neighbors of v have already dropped out. In this case, v will inevitably complete the inner for loop, and set *status* = `in-MIS`. An *early round* is a Bitty phase in which v is active, and at least one neighbor of v is also active. Note that every round in which v spends energy within a Luby phase is either early or late, but not both.

Since all the late rounds must occur within a single Luby phase, these must contribute at most $\beta \log n$ to v's energy expenditure.

We now examine the early rounds one by one, always conditioning on the outcomes of all previous rounds. Say that an early round is *fruitful* if v's bit of x for that round is a 0, and at least one active neighbor's bit for that round is a 1. Since there is at least one neighbor, and the coin flips are independent, every early round has probability at least $1/4$ to be fruitful, regardless of the prior history. Consider the first $8C \log n$ early rounds. In expectation, at least $2C \log n$ of them are fruitful. Applying Chernoff's bound for the lower tail, with $\delta = 1/2$, we have

$$\mathbb{P}(\text{number of fruitful rounds} < C \log n) < \exp\left(-\frac{(1/2)^2}{2} \cdot 2C \log n\right) < \frac{1}{n^2},$$

where the last inequality holds for large enough constant C.

Hence, with probability at least $1 - 1/n^2$, in the first $8C \log n$ early rounds there are at least $C \log n$ fruitful rounds. But, since each fruitful round causes v to drop out of the current Luby phase, and there are at most $C \log n$ Luby phases, it follows that v is active for at most $8C \log n$ early rounds. Adding these to the at most $\beta \log n$ late rounds (discussed earlier), and at most $C \log n$ rounds in between Luby phases, corresponding to lines 13 or 17 in the pseudocode, our final upper bound on energy spent is $(9C + \beta) \log n = O(\log n)$.

A union bound over all vertices v shows that, with probability at least $1 - 1/n$, no vertex spends more than $O(\log n)$ energy, completing the proof. ◀

Proof of Lemma 17. We prove by induction on $i \in [1, C\log n + 1]$ the following claim: at the start of any Luby phase i, the nodes with status `in-MIS` form an independent set with probability at least $1 - 9(i-1)/n^2$. (For $i = C\log n + 1$, you can consider instead the end of Luby phase $C\log n$.)

For the base case of $i = 1$, the statement holds trivially and with probability 1 since no node has status `in-MIS`. Now, we consider some Luby phase $i \geq 1$, such that the nodes with status `in-MIS` at its start, denoted by M_i, is independent (with probability at least $1 - 9(i-1)/n^2$). Note that in Luby phase i, only nodes having joined $W_i \cup C_i$ during the competition may change their status to `in-MIS`. Out of these, nodes in W_i form an independent set with probability at least $1 - 6/n^2$ (by Lemma 15 and a union bound over all nodes pairs). Moreover, by Lemma 9 (and our choice of C'), any node in W_i detects if it has a neighbor in M_i with probability at least $1 - 1/n^5$, in which case it does not set its status to `in-MIS`. By a simple union bound, with probability at least $1 - 1/n^4$, all nodes in W_i detects whether it has a neighbor in M_i. Then, if we let W_i^* denote those nodes of W_i that joined the MIS, then $M_i \cup W_i^*$ is independent with probability at least $1 - 6/n^2 - 1/n^4$.

Next, we consider the nodes of C_i that change their status to `in-MIS`. Note that any node in C_i detects if it has a neighbor in $M_i \cup W_i^*$ with probability at least $1 - 1/n^5$, in which case it does not set its status to `in-MIS`. Once again, we can apply a union bound, but now to the nodes of C_i. It remains to consider nodes in C_i^* – defined as the nodes of C_i that did not detect any MIS neighbors – because C_i^* may contain some adjacent nodes. However, nodes in C_i^* run LowDegreeMIS on an induced subgraph of maximum degree $O(\log n)$ (by Corollary 13) and hence by applying Davies' algorithm (cf. Section 4.2), nodes in C_i^* that have status `in-MIS` (denoted here by M_i') form an independent set with probability at least $1 - 1/n^2$. Therefore, it follows that the set of all nodes that have status `in-MIS` by the end of Luby phase i (and thus at the start of phase $i+1$), which is $M_i \cup W_i^* \cup M_i'$, is independent with probability at least $1 - 9(i-1)/n^2 - (6/n^2 + 2/n^4 + 1/n^2) \geq 1 - i \cdot (9/n^2)$. This completes the induction step, and the lemma statement follows when we consider $i \leq C\log n + 1$. ◀

To prove the following two lemmas, we will need the following definitions. For any Luby phase i, let D_i^{start} denote all non-terminated, MIS-dominated nodes at the start of the phase, and D_i denote all such nodes immediately prior to the shallow check of that phase (i.e., in Line 23 of Algorithm 2). Moreover, let X_i denote the number of edges in the residual graph (i.e., in E_i) incident to D_i, and for any $v \in D_i$, let $X_i(v)$ denote the number of such edges incident to v.

Proof of Lemma 19. Let us denote by $N_i(v) = N(v) \cap V_{i-1}$ the neighbors of v within the residual graph, and within those, by $N_D(v) = N_i(v) \cap D_i^{start}$ those that start the Luby phase as MIS-dominated, and by $N_F(v) = N_i(v) \setminus N_D(v)$ those that do not.

Now, consider some arbitrary node v. First, note that any neighboring node $u \in N_F(v)$ that enters the MIS implies that v is in D_i (immediately prior to the shallow check) and thus that $X_i(v) \geq |N_i(v)|$. Second, if no neighbors of v (nor v itself) enters the MIS, then it still holds that $X_i(v) \geq |N_D(v)|$ since $D_i^{start} \subseteq D_i$ (because nodes in the MIS never change their status). Hence, we have

$$\mathbb{E}\left(X_i(v) \mid E_{i-1}\right) \geq |N_D(v)| + \mathbb{P}\left(v \in D_i \mid E_{i-1}\right) |N_F(v)|$$

Next, we lower bound the probability that $v \in D_i$, conditioned on E_{i-1}. We say that some neighbor $u \in N_F(v)$ is eligible with respect to v, and denote as E_u the corresponding event, if $u \in N_F(v)$ chooses a maximum bitstring over $N_i(v) \cup N_i(u)$ in the competition of this Luby phase. (For the sake of this analysis, we assume nodes with status `in-MIS` also choose

a bitstring – contrary to the algorithm description – but that each such bitstring is smaller than any bitstring chosen by any node without an `in-MIS` status.) Note that E_u implies that u chose a locally maximum bitstring, thus by Lemma 14, u sets its status to `win` after the competition of this Luby phase with probability at least $1 - 1/n^2$. Moreover, by definition of D_i^{start}, u has no neighbors with status `in-MIS` during the competition and the first deep check, so by the algorithm description, u sets its status to `in-MIS` in this Luby phase, prior to Line 23. In summary, E_u implies that v is in D_i with probability $1 - 1/n^2$. Moreover, note that the E_u events are mutually exclusive over $N_F(v)$, and $\mathbb{P}(E_u \mid E_{i-1}) \geq 1/(|N_i(v)| + |N_i(u)|)$. Hence, we have that

$$\mathbb{P}(v \in D_i \mid E_{i-1}) \geq \sum_{u \in N_F(v)} \mathbb{P}\left(E_u \mid E_{i-1}\right)\left(1 - \frac{1}{n^2}\right) \geq \left(1 - \frac{1}{n^2}\right) \sum_{u \in N_F(v)} \frac{1}{|N_i(v)| + |N_i(u)|}$$

Finally, we shall lower bound the expectation of X_i conditioned on E_{i-1}. By linearity of conditional expectation, and because we count each edge of X_i twice when summing $X_i(v)$ over all nodes $v \in V_{i-1}$, we have that

$$\begin{aligned}\mathbb{E}\left(X_i \mid E_{i-1}\right) &= \frac{1}{2} \sum_{v \in V_{i-1}} \mathbb{E}\left(X_i(v) \mid E_{i-1}\right) \\ &\geq \frac{1}{2} \sum_{v \in V_{i-1}} |N_D(v)| + \frac{1}{2}\left(1 - \frac{1}{n^2}\right) \sum_{v \in V_{i-1}} \sum_{u \in N_F(v)} \frac{|N_F(v)|}{(|N_i(v)| + |N_i(u)|}\end{aligned}$$

Let us denote by E_{i-1}^D all edges of E_{i-1} with at least one endpoint in D_i. Then,

$$\frac{1}{2} \sum_{v \in V_{i-1}} |N_D(v)| \geq |E_{i-1}^D|/4 + \sum_{v \in V_{i-1}} |N_D(v)|/4$$

Next, let us denote by V_{i-1}^+ all nodes in V_{i-1} for which $|N_D(v)| \geq |N_i(v)|/2$, and by E_{i-1}^+ all edges of E_{i-1} with at least one endpoint in V_{i-1}^+. Then,

$$\sum_{v \in V_{i-1}} |N_D(v)|/4 \geq \sum_{v \in V_{i-1}^+} |N_i(v)|/8 \geq |E_{i-1}^+|/8$$

Finally, if we define $E_{i-1}^R = E_{i-1} \setminus (E_{i-1}^+ \cup E_{i-1}^D)$, then we can reorder the double sum and ignore some terms to get, for the last term, that

$$\begin{aligned}\frac{1}{2}\left(1 - \frac{1}{n^2}\right) \sum_{v \in V_{i-1}} \sum_{u \in N_F(v)} \frac{|N_F(v)|}{(|N_i(v)| + |N_i(u)|} &\geq \frac{1}{2}\left(1 - \frac{1}{n^2}\right) \sum_{\{u,v\} \in E_{i-1}^R} \frac{|N_F(u)| + |N_F(v)|}{|N_i(v)| + |N_i(u)|} \\ &\geq \frac{1}{4}\left(1 - \frac{1}{n^2}\right) |E_{i-1}^R| \geq |E_{i-1}^R|/5\end{aligned}$$

where the last inequality holds for large enough n. It follows that

$$\mathbb{E}\left(X_i \mid E_{i-1}\right) \geq |E_{i-1}|/8. \quad \blacktriangleleft$$

Proof of Lemma 20. Consider any Luby phase i. By Lemma 9, every node in D_i detects the presence of a neighbor with status `in-MIS` with probability at least 1/8. If that happens, then that node sets its status to `out-MIS` (and terminates) by the end of the Luby phase and thus every of its incident edge leaves the residual graph. In other words, every edge incident on D_i leaves the residual graph (i.e., is not in E_i) with probability at least 1/8. There are X_i such edges, and by Lemma 19, $\mathbb{E}\left(X_i \mid E_{i-1}\right) \geq \frac{|E_{i-1}|}{8}$. Hence, it follows that $\mathbb{E}\left(|E_i| \mid E_{i-1}\right) \leq (1 - \frac{1}{64})|E_{i-1}|$. $\blacktriangleleft$

C Energy-Efficient Backoff Procedures

Designing algorithms in the no-CD model can be significantly more difficult than in the CD model. In particular, a crucial difference is that in the no-CD model, nodes can no longer distinguish silence from collisions. Hence, nodes work with less information than they otherwise would have. In particular, the only way for a node to determine whether one of its neighbors is sending is for *exactly one* of its neighbors to send while it listens.

A generic way to achieve this is via exponential backoff. (This is a well-known procedure, and is also referred to as Decay in some works.) At a high-level, nodes decide to take either a *sender* or *receiver* role for the entire backoff protocol, and the protocol ensures that any receiver that has at least one sender neighbor hears a message with constant probability [5]. More concretely, traditional backoff works as follows. In a first round, all sender nodes send a message, while receiving nodes simply listen. Then, each sender node flips a fair coin to decide whether to send again in the next round (say when flipping 1) or drop out of the backoff (when flipping 0). This repeats for $O(\log \Delta)$ rounds. These iterations of $O(\log \Delta)$ rounds can themselves be repeated, say up to k times, to boost the success probability – this follows from well-known statements on exponential backoff, but see also Lemma 9. And if we take k large enough, say $\Theta(\log n)$, then the success is guaranteed with high probability.

In the above (traditional) exponential backoff, all nodes must be awake in all $O(\log n \log \Delta)$ rounds. In contrast, we give energy-efficient adaptations of the traditional exponential backoff procedures. The sender-side backoff is modified so that senders transmit only once per iteration, leading to a guaranteed and significant energy efficiency for senders. The receiver-side backoff is modified so that once a node hears a message, it sleeps for the remainder of the backoff (i.e., essentially an energy-motivated early "termination"). Note that any receiver node with no sender neighbor will be awake throughout the entire backoff, while any receiver node with at least one sender neighbor will save a significant amount of energy in expectation. More concretely, in the latter case, such a receiver node will be awake in expectation for only a constant number of iterations before it hears a message and sleeps.

Note that senders and receivers have asymmetric energy complexities (captured in the below lemma), that is, senders use a logarithmic factor less energy than receivers. This asymmetry is a crucial factor in the low energy complexity of our MIS algorithm in Section 5. These energy-efficient backoffs also provide the same correctness guarantees as the non-energy-efficient ones.

Proof of Lemma 8. The round complexity upper bounds follow from the fact that both backoff procedures execute k iterations, each taking $O(\log \Delta)$ rounds. As for the awake complexity upper bounds, they follow from the fact that nodes transmit once per outer loop iteration in Snd-EBackoff and listen at most $O(\log \Delta_{est})$ times per outer loop iteration in Rec-EBackoff. ◀

Proof of Lemma 9. Consider a receiver node v, and at most Δ_{est} sender nodes neighboring v. First, note that these sender nodes participate in all backoff iterations, thus any receiver node has the same number of sender neighbors throughout all backoff iterations.

Moreover, the following claim holds for each backoff iteration: if (receiver) node v has at least one sender neighbor, then during that iteration v hears a message with probability at least $1/8$. Indeed, for any single backoff iteration, each sender chooses to transmit in round $j < \lceil \log \Delta \rceil$ with probability $1/2^j$, and in round $\lceil \log \Delta \rceil$ with probability $1/2^{\lceil \log \Delta \rceil - 1}$ (due to the capping). Let $2 \leq d_S(v) \leq \Delta_{est}$ be the number of sender neighbors of v. Note that if $d_S(v) = 1$, the lemma holds trivially. Then, in round $j = \lceil \log d_S(v) \rceil$, for which v is awake

Algorithm 4 Energy-efficient k-Repeated Backoff Procedures.

```
1:  procedure Snd-EBackoff(k, Δ)
2:      ▷ Senders send once per iteration, and any listener hears a sender neighbor (if one
          exists) with constant probability per iteration. ◁
3:      for i ← 1 to k do
4:          x ← Sample from a geometric distribution with parameter 1/2
5:          x ← min(x, ⌈log Δ⌉)
6:          for j ← 1 to ⌈log Δ⌉ do
7:              if j = x then
8:                  transmit 1
9:              else
10:                 sleep
11:
12: procedure Rec-EBackoff(k, Δ, Δ_est = Δ)
13:     ▷ The third argument is optional: when not specified, it defaults to Δ. ◁
14:     heard ← False
15:     for i ← 1 to k do
16:         ▷ While they have not yet heard a message, receivers listen for log of their
              approximate degree rounds per iteration. ◁
17:         for j ← 1 to ⌈log Δ⌉ do
18:             if not heard and j ≤ ⌈log Δ_est⌉ then
19:                 listen
20:                 if heard 1 then
21:                     heard ← True
22:             else
23:                 sleep
24:     return heard
```

and $1/2^j \in [1/(2d_S(v)), 1/d_S(v)]$, the probability that there is exactly one sender neighbor of v transmitting in round j is (by summing the probabilities of the mutually exclusive events that a given sender neighbor transmits alone) at least

$$\sum_{c=1}^{d_S(v)} \frac{1}{2^j}\left(1-\frac{1}{2^j}\right)^{d_S(v)-1} \geq d_S(v)\cdot\frac{1}{2d_S(v)}\left(1-\frac{1}{d_S(v)}\right)^{d_S(v)-1} \geq \frac{1}{8}$$

where the last inequality follows from $1-x \geq (1/4)^x$ for $x \in [0, \frac{1}{2}]$. Since v listens in all rounds (unless it has already heard a message), v hears a message in round j or earlier.

Finally, since the randomness used by the successive backoff iterations are independent, it follows that v does not hear any of its sender neighbor in k backoff iterations with probability at most $(7/8)^k$, or equivalently, learns that it has at least one sender neighbor with probability at least $1-(7/8)^k$. ◀

Asynchronous Latency and Fast Atomic Snapshot

João Paulo Bezerra
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Luciano Freitas
Nomadic Labs, Paris, France

Petr Kuznetsov
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Matthieu Rambaud
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Abstract

This paper introduces a novel, *fast* atomic-snapshot protocol for asynchronous message-passing systems. In the process of defining what "fast" means exactly, we spot a few interesting issues that arise when conventional time metrics are applied to long-lived asynchronous algorithms. We reveal some gaps in latency claims made in earlier work on snapshot algorithms, which hamper their comparative time-complexity analysis. We then come up with a new unifying time-complexity metric that captures the latency of an operation in an asynchronous, long-lived implementation. This allows us to formally grasp latency improvements of our atomic-snapshot algorithm with respect to the state-of-the-art protocols: optimal latency in fault-free runs without contention, short constant latency in fault-free runs with contention, the worst-case latency proportional to the number of active concurrent failures, and constant *amortized* latency.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Asynchronous systems, time complexity, atomic snapshot, crash faults

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.15

Related Version *Extended Version*: `https://arxiv.org/abs/2408.02562` [11]

Funding *João Paulo Bezerra*: Mazars, TrustShare Innovation Chair.
Petr Kuznetsov: CHIST-ERA-22-SPiDDS-05 (REDONDA project) and Agence Nationale de la Recherche (ANR-23-CHR4-0009).

1 Introduction

The *distributed snapshot* abstraction [14, 26] allows us to determine a consistent view of the global system state. Originally proposed in the asynchronous fault-free message-passing context, it was later cast to shared-memory models [3] as a vector of shared variables, exporting an *update* operation that writes to one of them and a *snapshot* operation that returns the current vector state. Atomic snapshot can be implemented from conventional read-write registers in a *wait-free* manner, i.e., tolerating unpredictable delays or failures of any number of processes. By applying the reduction from shared memory to message-passing [6], one can get an asynchronous distributed atomic-snapshot implementation that tolerates up to a minority of faulty processes. The *atomic-snapshot object* (ASO) is, in a strong sense, equivalent to *lattice agreement* (LA) [8, 17][1]: one can implement the other with no time overhead. A long line of results improve time and space complexities of ASO and LA algorithms in shared-memory [5, 4, 7] and message-passing [17, 21, 19, 16, 18] models.

[1] Lattice agreement can be seen as a weak version of consensus, where decided values form totally ordered *joins* of proposed values in a *join semi-lattice*.


© João Paulo Bezerra, Luciano Freitas, Petr Kuznetsov, and Matthieu Rambaud;
licensed under Creative Commons License CC-BY 4.0



Table 1 Comparative time complexity of atomic-snapshot algorithms in asynchronous message-passing models. The table shows results for Single-Writer Multi-Reader (**SWMR**) implementations.

	Fault-free w/o contention	Fault-free w/ contention	Worst-case	Amortized constant
Faleiro et al. [17]	2	16	$O(k)$	yes
Imbs et al. [21]	2	$O(n)$	$O(n)$	no
Garg et al. [18]	≥ 6	≥ 8	$O(k)$	yes
Garg et al. [18] + Zheng et al. [27]	$O(\log n)$	$O(\log n)$	$O(\log n)$	no
Delporte et al. [16]	2	$O(n)$	$O(n)$	no
This paper	2	8	$O(k)$	yes

In this paper, we focus on the *latency* of operations in message-passing ASO implementations. We propose an LA (and, thus, ASO) algorithm that is *faster* than (or matches) state-of-the-art solutions in all execution scenarios: with or without failures and with or without contention. The comparative analysis of our algorithm with respect to the existing work appeared to be challenging: as we show, earlier work considered diverging metrics and execution scenarios, and sometimes used over-simplified reasoning. We observed that conventional metrics [13, 6, 2] are not always suitable for *long-lived* asynchronous algorithms. Besides, prior latency analyses of ASO and LA algorithms [17, 21, 19, 16, 18] used different ways to measure time, which complicated the comparison. We therefore propose a unifying time-complexity analysis of prior asynchronous ASO and LA algorithms with respect to a new metric, which we take as a contribution on its own.

Lamport [25] proposed to measure time in asynchronous systems as the length of the longest chain of *causally related* messages, the metric used to to determine the best-case latency of consensus [25] and Crusader Agreement [1]. However, as we show in this paper, the metric may produce counter-intuitive results for protocols involving all-to-all communication. For instance, in the failure-free case, n-process reliable broadcast [12] exhibits a causal chain of n hops, even though, intuitively one expects it to terminate in one.

Building upon the classical approach by Canetti and Rabin [13], Abraham et al. [2] recently proposed an elegant metric to grasp the good-case latency of broadcast protocols. We observe, however, that the metric does not really apply to executions of *long-lived* abstractions, which may contain *holes* – periods of inactivity when no protocol messages are in transit. Moreover, we get diverging results when applying [2] and [13] to *operation latency*, i.e., the time between invocation and response events of a given operation.

We therefore extend the round-based approach to long-lived abstractions (such as ASO and LA) and establish a framework to measure the time between arbitrary events, subsequently showing that the results align with those from earlier classical metrics [6, 13].

To summarize, our main contribution is a novel LA (and, thus, ASO) protocol that is generally *faster* than prior solutions, i.e., it exhibits shorter latency of its operations in various scenarios. In our complexity analysis, we compared our protocol to the original long-lived LA algorithm by Faleiro et al. [17][2], the first direct message-passing ASO implementation by Delporte et al. [16], the ASO algorithm based on the *set-constraint broadcast* by Imbs et al. [21], and the ASO algorithms by Garg et al. based on generic construction of ASO from *one-shot* LA with constant latency in fault-free runs [18] or $\log n$ worst-case latency by Zheng et al. [27] (where n is the number of processes).

[2] We consider the ASO protocol built atop the lattice agreement protocol proposed in [17].

As shown in Table 1, in a fault-free run, the latency of an operation of our protocol is the optimal two rounds if there is no contention and eight rounds in the presence of contention (four rounds if we ignore the "buffering" period when a value is submitted but not yet proposed), regardless of the number of contending operations. Moreover, the worst-case latency of our algorithm is proportional to the number of *active failures* k, i.e., the number of faulty processes whose messages are received within the operation's interval, therefore the *amortized* latency (averaged over a large number of operations in a long-lived execution) converges to the fault-free constant.

Our protocol can be seen as a novel combination of techniques employed separately in prior work. These include the use of *generalized* (long-lived) lattice agreement as a basis for ASO [23], the *helping* mechanism where all the learned lattice values are shared [23], relaying of messages to all replicas instead of quorum-based rounds [21, 18, 22, 15], and buffering proposed values until previous proposals get committed [17]. Similar to earlier proposals [17], our algorithm involves $O(n^2)$ (all-to-all) communication, which is compensated by its constant (amortized) latency. An interesting open question is whether one can reduce the communication cost in good runs, while maintaining constant amortized latency.

The paper is organized as follows. In Section 2, we present our model assumptions, and in Section 3, we state the problem of atomic snapshot and relate it to generalized lattice agreement. In Section 4, we present our protocol and analyze its correctness. In Section 5, we discuss several gaps in the complexity analyses of earlier work. In Section 6, we present a comparative analysis of time metrics. Certain proofs and a detailed discussion of time complexity of earlier protocols are delegated to the appendix.

2 System Model

Processes and Channels. We consider a system of n *processes* (or *nodes*). Processes communicate by exchanging messages $m = (s, r, data)$ with a sender s, a receiver r, and a message content *data*.

A process is an automaton modeled as a tuple $(\mathcal{I}, \mathcal{O}, \mathcal{Q}, q_0, \pi)$, where $\mathcal{I}$ is a set of inputs (messages and application calls) it can receive, $\mathcal{O}$ is a set of outputs (messages and application responses), $\mathcal{Q}$ is a (potentially infinite) set of possible internal states, $q_0 \in \mathcal{Q}$ is an initial state and $\pi : 2^{\mathcal{I}} \times \mathcal{Q} \to 2^{\mathcal{O}} \times \mathcal{Q}$ is a transition function mapping a set of inputs and a state to a set of outputs and a new state. Each process i is assigned an algorithm A_i which defines $(\mathcal{I}, \mathcal{O}, \mathcal{Q}, q_0, \pi)$, a *distributed algorithm* is an array $[A_1, ..., A_n]$.

Events and Configurations. Application calls and responses are tuples $(i, aReq)$ and $(i, aRep)$ with a process identifier, a request, and a reply respectively.

An *event* e is a tuple (R, P, S) where R is a set of received messages and/or application calls, P is the set of nodes producing the event and S is a set of messages sent and/or application responses. We denote $\mathsf{receive}(e)$ as the set of messages received in the event, conversely, $\mathsf{send}(e)$ is the set of messages sent. A *message hop* is a pair (e, e') in which e' receives at least one message that was sent in e.

Messages in transit are stored in the *message buffer*.[3] A *configuration* C is an $(n+1)$-array $[M, s_1, ..., s_n]$ with the buffer's state $M = C[0]$ and the local state $s_i = C[i]$ of each node i $(i = 1, \ldots, n)$. Let C_0 denote the *initial configuration* in which every s_i is an initial state and the buffer M is empty.

[3] We assume that every message in the message buffer is unique.

Executions. An *execution* (or *run*) is an alternating sequence $C_0e_1C_1e_2...$ of configurations and events, where for each $j > 0$ and $i = 1, \ldots, n$:

1. $\mathsf{receive}(e_j) \subseteq C_{j-1}[0]$;
2. $e_j.S$ consists of messages and application outputs that the nodes in $e_j.P$ produce, given their algorithms, their states in C_{j-1} and their inputs in $e_j.R$; the nodes in $e_j.P$ carry their states from C_{j-1} to C_j, accordingly;
3. for the nodes $i \notin e_j.P$, $C_{j-1}[i] = C_j[i]$.

Each triple $C_{j-1}e_jC_j$ is called a *step*. In this paper, we consider algorithms defined by deterministic automata, and we assume a default initial state. Thus, we sometimes skip configurations and simply write $e_1e_2 \ldots$.

In an *infinite* execution, a process is *correct* if it takes part in infinitely many steps, and *faulty* otherwise. We only consider infinite executions in which $f < n/2$, where f is the number of faulty processes and n is the total number of processes. Moreover, in an infinite execution, messages exchanged among correct processes are eventually received, i.e., if there is an event e from a correct process sending a message m to another correct process, then there is e' succeeding e such that $m \in \mathsf{receive}(e')$.

We also assume that the communication channels neither alter nor create messages. Finally, we assume that the channels are *FIFO*: messages from a given source to a given destination arrive in the order they were sent. A FIFO channel can be implemented by attaching sequence numbers to messages, without extra communication or time overhead.

3 Lattice Agreement and Atomic Snapshot

3.1 Lattice Agreement

A *join semi-lattice* is defined as a tuple $(\mathcal{L}, \sqsubseteq)$, where $\sqsubseteq$ is a partial order on a set $\mathcal{L}$, such that for any pair of values u and v in $\mathcal{L}$, there exists a unique *least upper bound* $u \sqcup v \in \mathcal{L}$ ($\sqcup$ is called the *join* operator). Also, u and v in $\mathcal{L}$ are said to be *comparable* if $u \sqsubseteq v \vee v \sqsubseteq u$.

The (generalized) *Lattice Agreement* abstraction LA [17] defined over $(\mathcal{L}, \sqsubseteq)$ can be accessed by every node with operation $\mathsf{Propose}(v)$, $v \in \mathcal{L}$ (we say that the node *proposes* v) which triggers the reply event $\mathsf{Learn}(w)$ (we say that the node *learns* w). Each node may invoke Propose any number of times but does so *sequentially*, that is, it initiates a new operation only after the previous one has returned.[4] The abstraction must satisfy:

▶ **Definition 1** (Lattice Agreement (LA)).

- **Validity.** *Any value learned by a node is the join of some set of proposed values that includes its last proposal.*
- **Stability.** *The values learned by any node increase monotonically, with respect to* $\sqsubseteq$.
- **Consistency.** *All values learned are comparable, with respect to* $\sqsubseteq$.
- **Liveness.** *If a correct node proposes* v*, it eventually learns a value* w.

[4] Following [23], without loss of generality, we slightly modified the conventional LA interface [8, 17] by introducing the explicit Propose operation that combine proposing and learning the values, the properties of the abstraction are adjusted accordingly.

3.2 Atomic Snapshot Object (ASO)

An atomic snapshot object (ASO) stores a vector of values $R = [r_1, ..., r_m]$ and exports two operations: update(i, v) and snapshot(). The update(i, v) operation writes the value v in $R[i]$ and returns OK, and snapshot() returns the entire vector R. An ASO implementation guarantees that every operation invoked by a correct process eventually completes. It also ensures that each of its operations appears to take effect in a single instance of time within its interval, i.e., it is *linearizable* [20].

We say that an ASO is *single writer* **SW** (resp. *multi writer* **MW**) if for each of its registers $R[i]$, only a single process can call update(i, v) (resp. every process can call update(i, v)). In this paper, we focus mostly on **SWMR** atomic snapshot objects. In Table 1 we give results only for **SWMR**. A **MWMR** ASO can be devised from **SWMR** by adding an additional "read" phase when updating values (see Section 3.3 for more details).

Next, we show that ASO can be implemented on top of LA with no additional overhead.

3.3 From LA to ASO

To implement a **SWMR** ASO on top of LA, we build upon the transformation algorithm introduced in [23]. We consider a partially ordered set $\mathcal{L}^*$ of $(m+n)$-vectors (recall that m is the size of the ASO vector and n is the number of nodes), defined as follows.

A vector position $\ell \in 1, \ldots, m$ is defined as a tuple $(w, v) \in R_\ell$, where v is an element of a value set V equipped with a total order $\leq^V$, and $w \in \mathbb{N}$ is the number of write operations on position ℓ. A total order on R_ℓ is defined in the natural way: for any two tuples $(w_1, v_1) \leq^{R_\ell} (w_2, v_2) \equiv (w_1 < w_2) \vee (w_1 = w_2 \wedge v_1 \leq^V v_2)$. For each process $i = 1, \ldots, n$, the vector position $m + i$ stores the number of snapshot operations executed by i.

The lattice $\mathcal{L}^*$ of $(m+n)$-position vectors is then the composition $R_1 \times \ldots \times R_m \times \mathbb{N}^n$. The partial order $\sqsubseteq^*$ on $\mathcal{L}^*$ is then naturally defined as the compositions of $<^{R_1} \times \ldots \times <^{R_m} \times \leq^n$. The composed join operator $\sqcup^*$ is the composition of max operators, one for each position in the $(m+n)$-position vectors. The construction implies a join semi-lattice [23].

In Algorithm 1, we show how to implement an **SWMR** atomic snapshot on top of LA defined over the semi-lattice $(\mathcal{L}^*, \sqsubseteq^*, \sqcup^*)$. For simplicity, we assume that $m = n$, i.e., the size of the array is the total number of nodes, and that each node i has a dedicated register i where it can write. Elements of $\mathcal{L}^*$ are then $2n$-vectors.

When a node i calls update(i,v), it increments its local *writing* sequence number w and proposes a $2n$-vector with (w, v) in position i and initial values in all other positions to the LA object. The vector learned from this proposal is ignored. When the node i calls snapshot(), it increments its local *reading* sequence number r proposes a $2n$-vector with r in position $n + i$ and initial values in all other positions to the LA object. The values in the first n positions of the returned vector is then returned as the snapshot outcome.

Algorithm 1 can be extended to implement a **MWMR** ASO: to update a position j in the array, a node first takes a snapshot to get the current state, gets up-to-date sequence number in position j and proposes its value with a higher sequence number. With this modification, the update operation takes two LA operations instead of one. We refer the reader to [23] for further details.

▶ **Theorem 2.** *Algorithm 1 implements ASO.*

Proof. See Appendix A. ◀

Algorithm 1 LA → **SWMR** ASO conversion.

```
1: Distributed objects:
2:     LA instance on (ℒ*, ⊑*, ⊔*)

3: upon startup
4:     w ← 0
5:     r ← 0

6: operation update(i, v)
7:     w ← w + 1
8:     V ← 2n-vector with (w, v) in position i and initial values in all other positions
9:     LA.Propose(V)

10: operation snapshot()
11:     r ← r + 1
12:     V ← 2n-vector with r in position n + i and initial values in all other positions
13:     return (LA.Propose(V))[1..n]
```

4 LA Protocol

In Algorithm 2, we describe our protocol for solving LA. To guarantee amortized constant complexity, the protocol relies on two basic mechanisms, employed separately in earlier work [17, 23]. First, when a node receives a request (e.g., a value from the application), it first adds the request to a buffer (*MPool*) and then relays it before starting a proposal. This ensures that "idle" nodes also help in committing the request. Second, the node relays every learned value so that nodes that are "stuck" can adopt values from other nodes.

4.1 Overview

The protocol is based on helping: every node tries to commit every proposed value it is aware of. As long as the node has *active* proposals that are not yet committed, it buffers newly arriving proposals in the local variable *MPool*. Intuitively, in the worst case, an LA.Propose operation has to wait until *one* of the concurrently invoked LA.Propose operations complete. Once this happens, the currently buffered value is put in the local dictionary *Pending* and shared with the other nodes (lines 31 and 32) via a **PROPOSE** message. In turn, the other nodes relay the message to each other (line 38). The dictionary maps a value to the number of times it is "supported" by the nodes (using **PROPOSE** messages). Once a value v in the dictionary assembles a quorum of $n-f$ of $\langle \mathbf{PROPOSE}, v \rangle$ messages, i.e., $Pending[v] \geq n-f$ (line 39), the value is added to the *Validated* variable. Once every value currently stored in *Pending* is in *Validated* (line 41), the operation completes with *Validated* as the learned value. As the final element of the helping mechanism, each process broadcasts every value it learns (lines 45 and 51), ensuring that processes that might otherwise remain "stuck" can complete their current proposal.

In summary, the algorithm relies on four main ideas: 1) buffering incoming requests when already proposing, 2) sharing every received proposal so all processes are quickly aware of active ones, 3) initiating a new proposal only after all currently seen proposals have been validated, and 4) broadcasting learned values to help other processes make progress.

Message Complexity. The protocol is comprised of three all-to-all communication phases: processes send and relay requests at lines 23 and 27, proposals at lines 32 and 38, and accepted values at lines 45 and 51. The total number of messages is therefore $O(n^2)$. However, a value in a **PROPOSE** message can include up to n distinct requests, and a value in a **ACCEPT** message may have arbitrary size. Therefore, in the extended version [11], we present a refined protocol description in which processes exchange $O(n^2)$ messages *per individual request*. This efficiency is achieved by relaying only the differences between current and previously received proposals and the learned values in phases 2 and 3, thus eliminating redundant messages with the same requests.

Algorithm 2 Long-Lived LA: code for node x.

14: **upon** Startup
15: $MPool, Proposing, Validated, Learned \leftarrow \bot$
16: $Pending \leftarrow \emptyset$

17: **operation** Propose(v)
18: SendRequest(v)
19: wait until $v \sqsubseteq Learned$
20: **return** $Learned$

21: **operation** SendRequest(v)
22: $MPool \leftarrow MPool \sqcup v$
23: **send** $\langle$**REQUEST**, $v\rangle$ **to** every other node

24: **upon** Receive $\langle$**REQUEST**, $v\rangle$ **from** a node
25: **if** $v \not\sqsubseteq MPool \sqcup Proposing \sqcup Learned$ **then**
26: $MPool \leftarrow MPool \sqcup v$
27: **send** $\langle$**REQUEST**, $v\rangle$ **to** every other node

28: **upon event** $(MPool \neq \bot) \wedge (Proposing = \bot)$
29: $Proposing \leftarrow MPool$
30: $MPool \leftarrow \bot$
31: $Pending[Proposing] \leftarrow 1$
32: **send** $\langle$**PROPOSE**, $Proposing\rangle$ **to** every other node

33: **upon** Receive $\langle$**PROPOSE**, $v\rangle$ **from** a node
34: **if** $v \in Pending$.keys() **then**
35: $Pending[v] + +$
36: **else**
37: $Pending[v] \leftarrow 1$
38: **send** $\langle$**PROPOSE**, $v\rangle$ **to** every node

39: **upon** exists v s.t. $Pending[v] = n - f$
40: $Validated \leftarrow Validated \sqcup v$

41: **upon event** $\bigsqcup Pending$.keys() $\sqsubseteq Validated$
42: **if** $Learned \sqsubset Validated$ **then**
43: $Learned \leftarrow Validated$
44: $Proposing \leftarrow \bot$
45: **send** $\langle$**ACCEPT**, $Learned\rangle$ **to** every node

46: **upon** Receive $\langle$**ACCEPT**, $w\rangle$ **from** a node
47: **if** $(Proposing \sqcup Learned \sqsubseteq w)$ **then**
48: $Validated \leftarrow Validated \sqcup w$
49: $Learned \leftarrow w$
50: $Proposing \leftarrow \bot$
51: **send** $\langle$**ACCEPT**, $Learned\rangle$ **to** every node

4.2 Correctness

Validity and Stability are immediate. We now proceed with Consistency and Liveness.

▶ **Lemma 3.** *If nodes i and j learn, resp., values w_i and w_j, then w_i and w_j are comparable.*

Proof. Suppose that $(w_i \not\sqsubseteq w_j) \wedge (w_j \not\sqsubseteq w_i)$. Then there must exist $v_i \sqsubseteq w_i$ and $v_j \sqsubseteq w_j$ such that $v_i \not\sqsubseteq w_j$ and $v_j \not\sqsubseteq w_i$.

Let Q_i (resp. Q_j) be the quorum i used to include v_i *Validated* at line 39. Since $Q_i \cap Q_j \neq \emptyset$, there is a common node x that sent $\langle \mathbf{PROPOSE}, v_i \rangle$ to i and $\langle \mathbf{PROPOSE}, v_j \rangle$ to j, but since channels are FIFO, either i received v_j or j received v_i from x before learning a value, therefore adding the value to *Pending*. Suppose it was i that received v_j before v_i, from the condition of line 41, i could not have learned w_i if $v_j \not\sqsubseteq$ *Validated*. ◀

▶ **Lemma 4.** *If a correct node x sets Proposing $= v$, x eventually learns a value with v.*

Proof. A node x sends a **PROPOSE** message to every other node whenever it adds a new value to *Pending* (line 38). If x is correct, it will receive at least $n - f$ **PROPOSE** messages for every value in *Pending*, adding the value to *Validated*. Therefore, the condition in line 41 is never satisfied from some point on only if x keeps adding a new value to *Pending* before all the current ones are validated.

Since each node proposes only one value at a time (until it learns a value, lines 28, 44, 50), for x to indefinitely add new values to *Pending*, there must be at least one other node that keeps learning values and proposing new ones. Without loss of generality, let y be one such node. Since faulty nodes eventually crash and stop taking steps, y must be correct. Every time y learns a new value w it sends $\langle \mathbf{ACCEPT}, w \rangle$ to x, and because channels are FIFO, x receives the **ACCEPT** message before the new value proposed by y. Eventually (because x sent its proposal to y), one of the received values w contains $x's$ *Proposing* and the condition on line 46 is satisfied, x then learns w. ◀

▶ **Lemma 5.** *If a correct node calls Propose(v), it eventually sets Proposing $= v'$, $v \sqsubseteq v'$.*

Proof. Let a correct node x call Propose(v), x then includes v in *MPool* (line 22). If x is not currently proposing, that is, the current value of *Proposing* is $\perp$, then it meets the condition in line 28 and immediately sets *Proposing* $=$ *MPool*. Otherwise, by Lemma 4, it eventually learns a value and sets *Proposing* $=\perp$ in lines 44 and 50, thus meeting the condition in line 28 and setting *Proposing* $=$ *MPool*. ◀

Lemmas 3, 4 and 5 imply:

▶ **Theorem 6.** *Algorithm 2 implements Generalized Lattice Agreement.*

▶ **Corollary 7.** *Algorithms 1 and 2 implement Atomic Snapshot.*

4.3 Time Metric

We now define the latency metric we are going to use in evaluating time complexity. Our metric is inspired by the metric proposed by Abraham et al. [2] (which in turn rephrases the original metric by Canetti and Rabin [13]). The distinguishing feature of our approach is that it also applies to *long-lived* executions and executions with *holes* (illustrated in Figure 1).[5]

Algorithm 3 describes the iterative method that assigns rounds to events in an execution. We give an informal description of the metric below.

[5] In Section 6, we show that the three metrics are equivalent in "hole-free" executions.

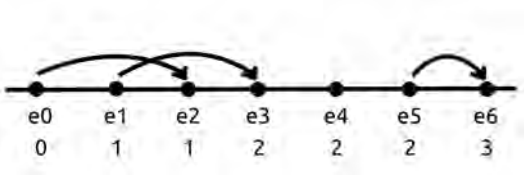


Figure 1 Example of round assignment using **IRA**. Arrows represent message transmissions and the number below an event corresponds to its round. A "hole" in communication appears betwen events e_3 and e_5.

▶ **Definition 8** (**I**terative **R**ound **A**ssignment - Informal)**.** *Algorithm 3 assigns round* 0 *to the initial event, and defines the end of round i as the last event that receives a message sent in round $i-1$. In addition, if there are no more messages to be received (or in transit), the event inherits the round number of its immediate predecessor.*

Algorithm 3 Iterative Round Assignment (**IRA**).

52: $e_0^* := e_0$
53: e_0 is assigned round 0
54: $r := 0$
55: **for** i=1... **do**
56: **if** e_i does not receive a message **then**
57: e_i is assigned round r
58: **else**
59: Let e_j be the oldest event from which e_i receives a message
60: Let r' be the round assigned to e_j ($r' \leq r$)
61: Let e' be the most recent event among $e_{r'}^*$ and e_j
62: All events after e' and up to e_i receive round $r'+1$
63: $e_{r'+1}^* := e_i$
64: $r = r'+1$

▶ **Definition 9** (**IRA** - Arbitrary Events)**.** *To measure the* latency between two events e_i and e_j*, we assign rounds according to Algorithm 3,* starting from e_i*, with* all events *up to and including e_i receiving round* 0*. The latency between e_i and e_j is then given by the round assigned to e_j.*

We say that an application request (or simply request, when there is no ambiguity) *completes* once the receiving node learns a value which includes the request. For a specific node i, we are interested in measuring the latency between the event e_C in which i receives a value v from the application software, and an event e_R, in which i learns a value w with v.

4.4 Time Complexity of Algorithm 2

We define latency as the number of rounds spanning between the moment a correct process receives an application call and the moment it returns from the operation. In evaluating the latency of our protocol, we consider two types of executions: (1) the *fault-free* case, when all processes are correct, and (2) the *worst-case*, when only a majority of processes are correct.

A snapshot operation *op precedes* another operation *op*′ if the response event of *op* happens before the call event for *op*′. Two operations are said to be *concurrent* if none precedes the other. For ASO protocols, we analyze latency in fault-free runs of an operation *op* in two distinct scenarios: (a) without contention, i.e., when no other operation overlaps in time with *op*, and (b) with contention, i.e., when there might be an arbitrary number of concurrent operations.

Garg et al. [19] use the notion of *amortized time complexity*, i.e., the average operation latency taken over a large number of operations in an execution. In some protocols, including ours, the latency of an operation is only affected by the number of faulty processes whose messages are received during the operation's interval (we call these processes *active-faulty*). Intuitively, faulty processes take a finite amount of steps, so in these protocols a failure can only affect a finite number of operations. In this paper, we also distinguish ASO protocols with *constant* time complexity. In Appendix B, we establish the optimality of our protocol under no-contention and prove the following results:

▶ **Theorem 10.** *An operation completes in at most* 2 *rounds in fault-free runs w/o contention.*

Consider an execution of our algorithm, and let F ($|F| \leq f$) be its set of faulty processes.

▶ **Theorem 11.** *An operation op takes at most* 8 *rounds to complete if, during its interval, no correct node receives a message from a faulty one.*

We say that there are k *active faulty nodes during an operation op* if, in between the call and return events for *op*, a message is received from a total of k distinct faulty nodes.

▶ **Theorem 12.** *An operation op takes $O(k)$ rounds to complete, where k is the number of active faulty nodes during op.*

▶ **Corollary 13.** *Algorithms 1 and 2 together have an amortized time complexity of* 8 *rounds.*

5 Measuring Latency of ASO Protocols

We conclude the paper with an overview of time complexity of earlier LA and ASO protocols [17, 16, 21, 18, 19]. We highlight certain gaps in their latency analyses and discuss the ways to fix them. Formalities and proofs can be found in the full version [11].

The First Message-Passing LA Protocol. Faleiro et al. [17] came up with the first LA implementation for asynchronous message-passing systems. They use the metric of [6] to measure latency and conclude that it takes $O(n)$ rounds to output from a lattice agreement operation in their protocol.

We show the somewhat surprising result that this protocol has constant latency of 16 rounds in fault-free runs. The upper bound holds as long as no message from faulty processes is received during the interval of the operation, implying that their LA protocol has constant amortized time complexity. We conjecture that the protocol has $O(k)$ worst-case latency, where k is the number of actual failures in the execution.

The First Direct ASO Implementation. Delporte et al. [16] is the first paper to directly implement ASO in message passing systems, instead of using an atomic register implementation [6] and the shared-memory snapshot construction [3].

In fault-free runs without contention, the latency of their protocol is only 2 rounds. In fault-free runs with contention, we support the claim of a bound of $O(n)$ rounds from [19].

ASO with SCD-Broadcast. Imbs et al. [21] introduce the abstraction of *Set Constrained Delivery Broadcast* (SCD − Broadcast), and show that it allows for implementing LA and ASO with no complexity overhead. In their complexity analysis, they assume bounded message delays and show that the latency of their ASO algorithm *in faulty-free and contention-free runs* is 2 rounds. We show that an operation of their resulting ASO algorithm can take $\Omega(n)$ rounds in fault-free runs *with contention*. We conjecture that this bound is tight, and so the time complexity of their ASO protocol is $\Theta(n)$.

A Generic ASO Algorithm. Garg et al. [18, 19] give a generic construction for atomic snapshot which uses any one-shot LA protocol as a building block (with constant latency overhead). The protocol thus inherits the asymptotic complexity of the underlying LA algorithm. They also provide a protocol for one-shot LA with 2 rounds latency in fault-free runs (using [13]'s metric). Their protocol requires 2 rounds of communication plus two lattice agreement invocations in the good case w/o contention and three lattice invocations with contention, making it at least 6 and 8 message delays, respectively.

For the worst-case latency analysis, they assume an additional requirement over communication channels: if a process executes $\mathsf{send}(m)$, sending m to a *correct* process, then m is eventually received (even if the sender is faulty). Using this assumption, they show a worst-case latency of $O(\sqrt{f})$ for their LA protocol.

In this paper, we assume a weaker channel that only guarantees delivery of messages among correct processes. We show that under this model, the LA protocol of [18] has an execution that takes $\Omega(f)$ rounds. We conjecture the upper bound of their protocol to be $O(f)$, and also that when using the stronger assumption, both our (Section 4) and [21]'s protocol have $O(\sqrt{f})$ worst-case latency.

The generic ASO construction may also be combined with the one-shot LA protocol presented in [27], which has worst-case latency of $O(\log f)$, providing an object whose update and snapshot operations take $O(\log f)$ in both fault-free fault-prone executions. For the sake of completeness, we also provide the time complexity analysis for the one-shot LA protocols from [17] and [21] in the full version [11].

6 Comparative Analysis of Time Measurement Metrics

In this section, we recall metrics used in the literature [6, 13, 2, 24] for measuring time in asynchronous systems. We exhibit executions where the metrics by Attiya et al. [6] and Canetti and Rabin [13] yield arbitrary results due to the presence of *holes* – "periods of silence" during which no messages are in transit – which are common in long-lived protocols. We show that in a subset of executions without holes, which we refer to as *covered executions*, these metrics align with the one proposed by Abraham et al. [2]. This is not surprising, as these metrics were designed for *distributed tasks*, which assume finite hole-free executions. We also recall Lamport's longest causal chain metric [24] and show that it is not suitable for comparing the ASO protocols we consider here.

Next, we show that the metric from [2] diverges from [6] and [13] when naïvely applied to measure time between arbitrary events. We then show that, after employing our refined method from Section 4.3 (Definition 9), they match when measuring rounds between arbitrary events in covered executions. Finally, we show that both our metric and that of [2] yield equivalent results in cases where [2] is applicable.

Altogether, we establish that our metric generalizes [2] and aligns with classical metrics [6, 13] when applied to distributed tasks. A summary of the comparative analysis is presented in Table 2. Proofs are provided in the extended version [11].

6.1 Definitions

Timed Executions. We assume a global clock, not accessible to the nodes. A *timed event* $\bar{e}$ is a pair (t, e) in which t is a non-negative real number, we also say that $\bar{e}$ is a *time assignment* of e. A *timed execution* is an alternating sequence $C_0\bar{e}_1C_1\ldots$ where $\bar{e}_1 = (t_1, e_1), \bar{e}_2 = (t_2, e_2), \ldots$, where events $e_1, e_2, \ldots$ are equipped with monotonically increasing times $t_1, t_2, \ldots$:

Table 2 Comparison between asynchronous time metrics. Metrics that are *timed* make use of time assignments to determine the number of rounds between events. We compare each metric against **CR**, evaluating the number of rounds resulting from applying them over entire (covered) executions and between arbitrary events. Blue stands for "good" features and red – for "bad" ones. The equivalence of NTR to CR holds as long as one uses Definition 9.

	Timed	Equivalent to **CR** (Covered Executions)	Equivalent to **CR** (Arbitrary Events)	Admits Holes
CR [13]	Yes	-	-	No
Round [6, 9]	Yes	Yes	Yes	No
NTR [2]	No	Yes	Yes	No
LCC [25]	No	No	No	Yes
IRA	No	Yes	Yes	Yes

1. $t_m > t_l$ whenever $m > l$;
2. $t_l \to \infty$ as $l \to \infty$.[6]

A *time assignment* of E is a timed execution $\overline{E}$ in which every event e_i in E is matched with a timed event (t_i, e_i) in $\overline{E}$ and the sequences of configurations in E and $\overline{E}$ are the same. Notice that an execution allows for infinitely many time assignments.

Let m be a message sent in $\overline{e}_l$ and received in $\overline{e}_m$, the *delay* of m is then defined as $t_m - t_l$. For a finite timed execution $\overline{E} = C_0\overline{e}_1 ... \overline{e}_l C_l$, we define $t_{start}(\overline{E}) = t_1$, $t_{end}(\overline{E}) = t_l$ (we use t_{start} and t_{end} when there is no ambiguity) and $duration(\overline{E}) = t_{end} - t_{start}$.

In the subsequent discussion, given an execution E, let $\mathcal{T}(E)$ denote the set of all timed executions $\overline{E}$ based on E.

Time Metrics. It is conventional to measure the execution time by the number of communication *rounds*, typically calculated using the "longest message delay." These metrics can be applied to both *executions* and *timed executions*. The first metric we consider is defined in Definition 14 [6]. When applied to timed executions, this metric assumes a known upper bound on message delays, which can be normalized to one time unit without loss of generality. To apply this metric to an *execution*, we consider the maximum duration of all possible timed executions that adhere to the upper-bound communication constraint.

▶ **Definition 14** (**Round** metric). *Given a timed execution $\overline{E}$, in which the maximum message delay is bounded by one unit of time, $\overline{E}$* takes *$duration(\overline{E})$* rounds.

By extension, an execution E takes $\sup_{\overline{E} \in \mathcal{T}(E)} duration(\overline{E})$ rounds.

In the metric proposed by Attiya and Welch [9, 10], the time assignments are scaled so that the maximum message delay is always 1, thus, the metric produces the same results for executions as Definition 14. A more general metric introduced by Canetti and Rabin [13] captures the time complexity of any finite execution. Let $\overline{E}$ be a timed execution, and let $\delta_{\overline{E}}$ be the maximum message delay in it. Then $\overline{E}$ *takes $duration(\overline{E})/\delta_{\overline{E}}$* **CR** *rounds.*

▶ **Definition 15** (**CR** metric). *A finite execution E takes $\sup_{\overline{E} \in \mathcal{T}(E)} duration(\overline{E})/\delta_{\overline{E}}$ rounds, where $\delta_{\overline{E}}$ is the maximum message delay of each corresponding timed execution.*

[6] We require this property to avoid the case where a never-terminating execution has a finite time duration.

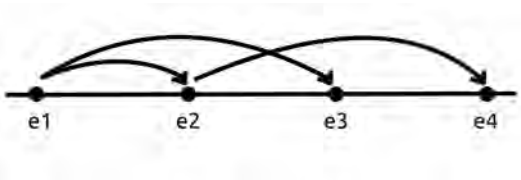


Figure 2 Example of an execution with 2 rounds in the Round, CR and NTR metrics.

▶ **Example 16.** Figure 2 shows an execution with four events, where we assign a delay of δ to the message exchanges (e_1, e_3) and (e_2, e_4), and a delay of $\delta - \epsilon$ $(\epsilon > 0)$ to (e_1, e_2). By making ϵ arbitrarily small, the number of rounds in this execution converges to 2 in the **CR** metric. The same result is obtained in the Round metric by setting $\delta = 1$.

Recently, Abraham et al. [2] proposed an approach that can be applied to executions without relying on time assignments. We call this metric **n**on-**t**imed **r**ounds (**NTR**):

▶ **Definition 17** (**NTR** metric). *Given an execution E, each event in E is assigned a round number as follows:*

- *The first event e_0 is assigned round 0. We also write $e_0^* = e_0$;*
- *For any $r \geq 1$, let e_r^* be the last event where a message of round $r-1$ is delivered. All events after e_{r-1}^* until (and including) event e_r^* are in round r.*

The number of rounds in E is the round assigned to its last event.

▶ **Example 18.** Coming back to Figure 2, if we assign a round to each event based on Definition 17 then e_1 gets round 0, e_2 and e_3 get round 1 and e_4 is assigned round 2. The execution has therefore 2 rounds according to **NTR**.

Lamport [25] proposed a metric for latency based on the causal chain of messages. The *Longest Causal Chain* (**LCC**) was used to show best-case latency of protocols such as consensus [25] and Crusader Agreement [1].

▶ **Definition 19** (**L**ongest **C**ausal **C**hain). *Let e be an event in E and M the set of messages received by e, then e is assigned round $k+1$, where k is the maximum round of an event originating a message in M. If $M = \emptyset$, then $k = 0$. The number of rounds in an execution becomes the highest round assigned to one of its events.*

This metric, however, diverges from **CR** and **NTR**.

▶ **Example 20** (Reliable Broadcast). In the *reliable broadcast* primitive [12], a dedicated source *broadcasts* a message and, if the source is correct, then all correct nodes should deliver the message. Furthermore, if a correct process delivers a message, then every correct process eventually delivers it. The following protocol satisfies this property:

- When the source invokes *broadcast(m)*, it delivers m and sends it to everyone;
- When a process receives m for the first time, it delivers m and sends it to everyone.

In Figure 3, we depict an execution of this protocol with four processes: p_1, p_2, p_3 and p_4. Here, p_1 is the source and broadcasts m, the message is received by p_2 which then sends m to everyone. Process p_3 receives m from p_2 before receiving it from p_1, and finally, p_4 receives m from p_1 in the last event. This execution has 2 **LCC** rounds, while having 1 round according to **CR** and **NTR**.

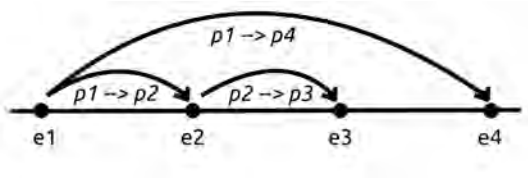
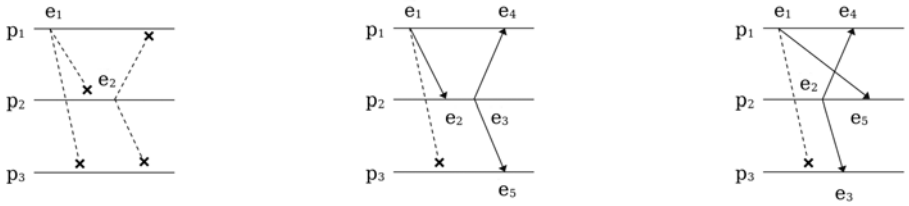


Figure 3 Example of a reliable broadcast protocol execution.



(a) Execution with undefined $\delta_{\overline{E}}$. **(b)** Execution where the number of rounds is unbounded according to Round and **CR**. **(c)** Covered execution.

Figure 4 Examples of non-covered and covered executions.

Example 20 shows that the **LCC** metric diverges from the others in cases where a fast exchange of messages happens in the interval of one slow message. This is the case for several ASO protocols in the literature (including ours) which heavily rely on relaying values to speed up the validation phase, making the metric unsuitable for our use case. On the other hand, **CR** and **NTR** provide equivalent results in *covered* executions, described next.[7]

6.2 Covered Executions and Holes

Consider an execution $E = C_0e_1C_1...e_lC_l$ illustrated in Figure 4a where no process receives a message from another process, i.e., events may add messages to the buffer but no event removes a message from it. $\delta_{\overline{E}}$ is not defined in any time assignment $\overline{E}$.

Now consider an execution $E' = C_0e_1C_1...e_lC_l...e_mC_m$ in which:

- A message m is sent in e_1 and received in e_l;
- A message m' is sent in e_{l+1} and received in e_m;
- No message from $e_1...e_l$ is received in $e_{l+1}...e_m$.

In this example, illustrated in Figure 4b with 5 events, $\delta_{\overline{E'}}$ exists for any time assignment of E', but we can still assign an arbitrary time difference to e_l and e_{l+1} without affecting $\delta_{\overline{E'}}$, which results in the number of **CR** rounds to be unbounded.

The two executions in the examples above have events whose time difference is unrelated to message delays. By consequence, the duration of these executions can grow irrespective of any bound imposed by message exchanges. Similarly, in Figure 4b, since there is no message being received in $e_3e_4e_5$ from e_1e_2, there is no round assignment defined when using **NTR** to e_3, e_4 and e_4.

We then restrict the analysis of these metrics to executions that are *covered*. Formally:

[7] The Round and **CR** metrics also provide equivalent results in covered executions [11].

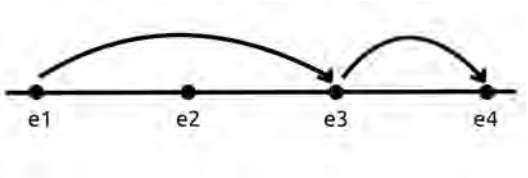


Figure 5 An execution in which there are 2 **CR** rounds between e_2 and e_4. However, the difference of the rounds assigned to e_2 and e_4 using **NTR** is 1.

▶ **Definition 21** (Covered Execution). *A* hole *in an execution is a pair* (e_l, e_{l+1}) *in which no event in* e_{l+1}... *receives a message from* ...e_l*, in other words, there are no message hops among the two sequence of events. An execution is* covered *iff it has no holes.*

Abraham et al. [2] introduce **NTR** as an equivalent to **CR**, however, no formal proof is provided. The next result corroborates this claim in covered executions. Later in Example 24, we show that using **NTR** naively to measure time between events may *not* match **CR**.

▶ **Theorem 22.** *A finite covered execution* E *has* k ***CR*** *rounds iff it has* $\lceil k \rceil$ ***NTR*** *rounds.*

6.3 Time Between Arbitrary Events

In long-lived executions (such as those of atomic snapshot algorithms) we are interested in measuring time between two events, for instance, between an application call and response. Definition 15 can easily be adapted to measure the number of rounds between two events as follows:

▶ **Definition 23** (Generalized **CR** metric). *Let* E *be an execution, let* $\mathcal{T}(E)$ *denote the set of all timed executions* $\overline{E}$ *based on* E*, and* $\delta_{\overline{E}}$ *- the maximum message delay in* $\overline{E}$*. Let* e_i *and* e_j $(j > i)$ *be events in* E*, and* t_i *and* t_j *time assignments in* $\overline{E}$ *for them respectively. Then we say that in between* e_i *and* e_j *there are:* $\sup_{\overline{E} \in \mathcal{T}(E)} (t_j - t_i)/\delta_{\overline{E}}$ **CR** rounds.

An appealing way of defining time between two events e_i and e_j using a non-timed metric is to assign rounds according to **NTR**, and then take the difference of rounds assigned to e_i and e_j. As illustrated in Example 24, this definition can diverge from generalized **CR**.

▶ **Example 24.** Consider the execution shown in Figure 5. We can assign times to e_1, e_3 and e_4 such that the two message hops have delay of δ. Now consider the number of rounds between e_2 and e_4, since we can assign a time for e_2 that is arbitrarily close to e_1's assignment, there are 2 **CR** rounds between e_2 and e_4. However, the round assignments using **NTR** to e_2 and e_4 are 1 and 2 respectively, so simply taking the difference between them leads to a value that diverges from **CR**.

We then give the following definition, using the approach described in Section 4.3:

▶ **Definition 25** (Generalized **NTR**). *Given an execution* E*, let* e_i *and* e_j $(j > i)$ *be events in* E*. The number of rounds between* e_i *and* e_j *is given by the round assigned to* e_j *according to the following:*

- *All events up to (and including)* e_i *are assigned round* 0*. We also write* $e_0^* = e_i$*;*
- *For any* $r \geq 1$*, let* e_r^* *be the last event where a message of round* $r - 1$ *is delivered. All events after* e_{r-1}^* *until (and including) event* e_r^* *are in round* r*.*

▶ **Theorem 26.** *Let* E *be a covered execution and* e_i *and* e_j $(j > i)$ *be events of* E*. There are* k *rounds in between* e_i *and* e_j *according to* ***CR*** *(Definition 23) iff there are* $\lceil k \rceil$ *rounds in between them according to* ***NTR*** *(Definition 25).*

6.4 Relating IRA to NTR

▶ **Theorem 27.** *Let E be a finite covered execution and suppose that all events of E are assigned rounds according to **IRA** after all iterations of the algorithm. It holds that:*

1. *Round 0 is composed only of e_0 (the initial event).*
2. *The final event of round $i+1$ is the last event to receive a message from round i.*

▶ **Corollary 28.** ***IRA** and **NTR** assign the same rounds to events in covered executions.*

References

1 Ittai Abraham, Naama Ben-David, Gilad Stern, and Sravya Yandamuri. On the round complexity of asynchronous crusader agreement. *Cryptology ePrint Archive*, 2023.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. *CoRR*, abs/2102.07240, 2021. arXiv:2102.07240.

3 Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, 1993. doi:10.1145/153724.153741.

4 James Aspnes, Hagit Attiya, Keren Censor-Hillel, and Faith Ellen. Limited-use atomic snapshots with polylogarithmic step complexity. *J. ACM*, 62(1):3:1–3:22, 2015. doi:10.1145/2732263.

5 James Aspnes and Keren Censor-Hillel. Atomic snapshots in o(log3 n) steps using randomized helping. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2013. doi:10.1007/978-3-642-41527-2_18.

6 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, January 1995. doi:10.1145/200836.200869.

7 Hagit Attiya, Faith Ellen, and Panagiota Fatourou. The complexity of updating snapshot objects. *J. Parallel Distributed Comput.*, 71(12):1570–1577, 2011. doi:10.1016/J.JPDC.2011.08.002.

8 Hagit Attiya, Maurice Herlihy, and Ophir Rachman. Atomic snapshots using lattice agreement. *Distributed Comput.*, 8(3):121–132, 1995. doi:10.1007/BF02242714.

9 Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

10 Hagit Attiya and Jennifer L Welch. Multi-valued connected consensus: A new perspective on crusader agreement and adopt-commit. In *27th International Conference on Principles of Distributed Systems*, 2024.

11 João Paulo Bezerra, Luciano Freitas, and Petr Kuznetsov. Asynchronous latency and fast atomic snapshot. *arXiv preprint arXiv:2408.02562*, 2024.

12 Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming.* Springer Science & Business Media, 2011.

13 Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993. doi:10.1145/167088.167105.

14 K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985. doi:10.1145/214451.214456.

15 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

16 Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, and Michel Raynal. Implementing snapshot objects on top of crash-prone asynchronous message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 29(9):2033–2045, 2018. doi:10.1109/TPDS.2018.2809551.

17 Jose M. Faleiro, Sriram Rajamani, Kaushik Rajan, G. Ramalingam, and Kapil Vaswani. Generalized lattice agreement. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 125–134, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2332432.2332458.

18 Vijay Garg, Saptaparni Kumar, Lewis Tseng, and Xiong Zheng. Amortized constant round atomic snapshot in message-passing systems. *arXiv preprint arXiv:2008.11837*, 2020. arXiv:2008.11837.

19 Vijay K. Garg, Saptaparni Kumar, Lewis Tseng, and Xiong Zheng. Fault-tolerant snapshot objects in message passing systems. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139, 2022. doi:10.1109/IPDPS53621.2022.00113.

20 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

21 Damien Imbs, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Set-constrained delivery broadcast: Definition, abstraction power, and computability limits. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018. doi:10.1145/3154273.3154296.

22 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

23 Petr Kuznetsov, Thibault Rieutord, and Sara Tucci-Piergiovanni. Reconfigurable Lattice Agreement and Applications. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*, volume 153 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2019.31.

24 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications*, 1978.

25 Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19:104–125, 2006. doi:10.1007/S00446-006-0155-X.

26 Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *J. Parallel Distributed Comput.*, 18(4):423–434, 1993. doi:10.1006/JPDC.1993.1075.

27 Xiong Zheng, Vijay K. Garg, and John Kaippallimalil. Linearizable Replicated State Machines With Lattice Agreement. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*, volume 153 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2019.29.

A Linearizability of Algorithm 1

The *history* of an execution E is the subsequence of E consisting of invocations and responses of ASO operations (update and snapshot). A history is *sequential* if each of its invocations is followed by a matching response. An execution is linearizable if, to each of its operation (update or snapshot, except, possibly, for incomplete ones), we can assign an indivisible point within its interval (called a *linearization point*), so that the operations put in the order of its linearizaton points constitute a *legal* sequential history of ASO (called a *linearization*), i.e., every snapshot operation returns a vector where every position contains the last value written to it (using an update operation), or the initial value if there are no such prior updates. Equivalently, a linearizable execution E with history H should have a linearization S, a legal sequential history that (1) no node can locally distinguish a completion of H and

S and (2) S respect the *real-time order* of H, i.e., if operation op completes before operation op' in H, then op' cannot precede op in S.

We show that every execution of Algorithm 1 is linearizable. Consider an execution of Algorithm 1, let H be its history. Every operation (snapshot or update) is associated with a unique sequence number and performs a Propose operation on the LA object. If there is an LA.Propose operation that returns (w, v) in position i, by Validity of LA, there is an operation update(i, v) executed by node i with sequence number w that started before the LA.Propose completed and invoked a LA.. In this case, we say that the update operation is *successful*. Notice that by Validity of LA, the update must have invoked LA.Propose with a vector containing (w, v) in position i.

Now we order complete snapshot operations and complete successful update operations in the order of the values returned by their LA.Propose operations (by Consistency of LA, these values are totally ordered. As each of these LA.Propose returns a value containing its unique sequence number (Stability of LA) , this order respects the real-time order of H. A successful update operation performed by node i with (w, v) in position i that has no complete LA.Propose is placed right before the first snapshot whose LA.Propose returns this value. By construction, the resulting sequential history is legal and locally indistinguishable from a completion of H.

Finally, Liveness implies that every operation invoked by a correct process eventually completes.

B Time Complexity of Algorithm 2

We establish the optimality of our protocol under no-contention. A protocol implementing LA *tolerates* k faults if it satisfies all the properties of Definition 1 in every execution with at most k faulty processes.

▶ **Theorem 29.** *Let $\mathcal{P}$ be a distributed protocol that implements LA and tolerates at least one faulty process. Then, there exists a fault-free run of $\mathcal{P}$ in which an LA operation requires at least two rounds of communication to complete without contention.*

Proof. Consider an operation op initiated by node x, with call event e_C and response event e_R. Suppose op completes in at most one round in fault-free, contention-free executions.

We first show that there exists an execution $E = e_1, \ldots, e_C, \ldots, e_R$ such that:

- x is the only process to take a step in e_R,
- no message sent by x in $e_C, \ldots, e_R$ is received by any other process before e_R.

If multiple processes perform steps in the same event e, we can conceptually "split" e into a sequence of events $e^1, e^2, \ldots$, where each process takes the step in its own dedicated event. Since their steps are independent, these split events are indistinguishable from the original e from each process's perspective. This reasoning also applies to e_R.

Now, assume for the sake of contradiction that in every fault-free, contention-free execution containing both e_C and e_R, there exists some process $y \neq x$ that receives a message m–sent by x in the interval $e_C, \ldots, e_R$–before e_R occurs.

Let e_M denote the event where y receives m. We define rounds from e_C's perspective:

- all events up to e_C are in round 0,
- round 1 ends at the last event e_L that receives a message originating in round 0.

If m is sent after e_C, then we can construct E so that all messages from round 0 are received before m. This ensures that e_M occurs after e_L, meaning e_M is in round 2. Since e_R

occurs after e_M, it too is assigned round 2–contradicting our assumption that *op* completes in one round.

If instead m is sent in e_C, we can again construct the execution so that all round 0 messages are received before or at the same time as m, making $e_M = e_L$. Since e_R occurs after e_M, it is again assigned to round 2–a contradiction.

These contradictions hold regardless of whether *op* is concurrent with any other operation. Hence, such an execution E must exist. Now consider an extension E' of E where all messages sent by x after (and including) e_C are indefinitely delayed, while messages from other nodes are not.

Suppose a node z invokes a new operation op' after e_R, making op' non-concurrent with *op*. Since protocol $\mathcal{P}$ tolerates at least one faulty process, and x appears to have crashed in E', node z must eventually complete op' without any process receiving any messages from x.

Let v and w be the value proposed and the value learned by x in *op*, and let v' and w' be the corresponding values for z in op'. By Validity, we know $v \sqsubseteq w$, and by Consistency, we know $w \sqsubseteq w'$, hence $v \sqsubseteq w'$.

However, since no process receives a message from x since e_C, no one could have known about v, contradicting the requirement that w' must contain v.

Finally, after op' completes, we can allow all delayed messages from x to be received, making all processes correct in the final execution E'. This completes the proof. ◀

▶ **Theorem 30.** *An operation completes in at most* 2 *rounds in fault-free runs w/o contention.*

Proof. Consider a contention-free request with call event e_C and return event e_R invoked by a node i. There are no call events for other nodes between e_C and e_R, but some messages from previous proposals may still be in transit.

Suppose v is the value to be proposed for the application call. If i is not proposing (has $Proposing = \perp$) when it receives v, then it directly sends $\langle \mathbf{PROPOSE}, v \rangle$ to everyone. Let e_P be the last event in which a process receives $\langle \mathbf{PROPOSE}, v \rangle$ from i, then every process also sends $\langle \mathbf{PROPOSE}, v \rangle$ by at most e_P. Now take e_F as the final event in which a process receives $\langle \mathbf{PROPOSE}, v \rangle$ in the execution, and e_S as the corresponding sending event. It must be that e_S happens between e_C and (potentially including) e_P. Also, because the channels are FIFO, every previous proposal must have been validated before e_F, and i will learn a value containing v by at most e_F. Let e_C be assigned round 0, then e_P happens at most in round 1. As a consequence, e_S is assigned either 0 or 1, thus e_F can be assigned at most round 2. Then, by the end of round 2, i already has v validated.

Now suppose that i is proposing when it receives v, so it still has a value v' in *Pending* that is not validated, w.l.o.g. assume that v' is the only one. This value must be from a call that already finished, and the corresponding node sent $\langle \mathbf{ACCEPT}, w \rangle$ containing v' before e_C. Consider two pairs of events: (e_A, e'_A) and (e_C, e'_C). In the first pair, e_A is the event where $\langle \mathbf{ACCEPT}, w \rangle$ was first sent, and e'_A is the *last* event in which $\langle \mathbf{ACCEPT}, w \rangle$ is received from e_A. In the second, e_C is the usual application call event and e'_C is the *last* event in which $\langle \mathbf{REQUEST}, v \rangle$ is received from i. There are two cases to consider: 1) e'_A happens before e'_C and 2) e'_C happens before e'_A.

If it is the first case, then at the moment e'_C happens, every node was already able to propose v (since there was no other value to be learned). Take the last event e_L in which a $\langle \mathbf{PROPOSE}, v \rangle$ (or a value containing v) is received, and e_S as the corresponding sending event, it follows that i validates v by at most e_L and can learn a value containing it. Let e_C be assigned round 0, e'_C and e_S can be assigned at most round 1, and since e_L receives a message from e_S, it can be assigned at most round 2. If it is the second case, then all

nodes received $\langle \mathbf{REQUEST}, v \rangle$ and put v in *MPool* before e'_A. Every node proposes v by at most e'_A (since they can adopt w and stop any current proposal). Let e_L be the last event in which a process receives a proposal for v and e_S it's corresponding sending event, similarly to the above cases, e_S happens between e_C and e'_A. Now, let e_A and e_C be assigned round 0. e_S can be assigned at most round 1 (e_S happens before or at e'_A) and e_L at most 2, which concludes the proof. ◀

▶ **Lemma 31.** *Consider an event in which a correct node sends $\langle \boldsymbol{PROPOSE}, v \rangle$ and the first event in which a correct node learns a value including v. If no correct node receives a message from a faulty one between these two events, then there are at most 3 rounds between them.*

Proof. A message sent by a correct node is received by every correct node in the execution, and since correct nodes do not receive messages from faulty ones in the interval we are analyzing, we can consider only events originated from correct nodes. Therefore, we only refer to correct nodes in the following.

Let x be the node sending $\langle \mathbf{PROPOSE}, v \rangle$, e_P be the corresponding event and e'_P the last event a node receives $\langle \mathbf{PROPOSE}, v \rangle$ from x. Because x also sends $\langle \mathbf{REQUEST}, v \rangle$, by e'_P every node received the request and must be proposing. Any value learned after e'_P contains v since all nodes have v in *Pending*.

Now, at the configuration just after applying e'_P, let V be the set in which $w \in V$ satisfies: there exists a (correct) node where w is in *Pending* but is not yet validated. Consider a value $w \in V$ that is the last whose $\langle \mathbf{PROPOSE}, w \rangle$ is received by any node, where e'_L is the event in which $\langle \mathbf{PROPOSE}, w \rangle$ is last received and e_L the corresponding sending event. It follows that some node learns a value containing v by at most e'_L.

Next, take the first event e_F in which a node sent $\langle \mathbf{PROPOSE}, w \rangle$, and e'_F the event in which the last $\langle \mathbf{PROPOSE}, w \rangle$ from e_F is received. Note that e_L happens at most at e'_F and e_F at most at e'_P. Let e_P be assigned round 0, then e'_P (and thus e_F) can be assigned at most round 1, e'_F (and thus e_L) at most 2 and lastly, e'_L can be assigned at most round 3. Therefore, there are at most 3 rounds between a propose and the first learn event for v. ◀

▶ **Theorem 32.** *An operation op takes at most 8 rounds to complete if, during its interval, no correct node receives a message from a faulty one.*

Proof. Let v be the value received from the application call for *op*, e be the event in which node i proposes v (or a value containing v) and e' the event in which a value including v is learned for the first time. From Lemma 31, there are at most 3 rounds between e and e'. Since the node that learns v sends $\langle \mathbf{ACCEPT}, v \rangle$ to everyone, i receives and adopts it in one extra round. We conclude that in at most 4 rounds every correct node can learn v.

If i is already proposing a value when it receives a call for v, it sends $\langle \mathbf{REQUEST}, v \rangle$ to everyone and put it in *MPool*, so it is proposed next. Let e_P be the event in which i initiated its previous proposal to v, and consider the worst case where the application call e_C with v happens just after e_P. From e_P to the event in which i learns its previous proposal e'_P (and thus starts proposing v), there are at most 4 rounds, and from e'_P to the learning event of v there are also at most 4 rounds. Therefore, the operation completes in at most 8 rounds. ◀

▶ **Theorem 33.** *An operation op takes $O(k)$ rounds to complete, where k is the number of active faulty nodes during op.*

Proof. We show that an operation *op* in Algorithm 2 takes $O(k)$ rounds to complete, where k is the number of active faulty nodes during *op*.

Messages from and to faulty nodes may not arrive, however, a message sent by (and to) a faulty node at round r is received at most by round $r+1$. Moreover, since channels are FIFO, when a node i receives a message from another node j, i must also have received all previous messages j sent to i, irrespective of them being correct or faulty.

If a correct node receives $\langle \textbf{PROPOSE}, v' \rangle$ (even from a faulty node) in round r, every correct node will have v' added to *Pending* by the end of round $r+1$, and will have v' validated by the end of round $r+2$. Also, faulty nodes wait for its current proposal to finish before starting a new one, in which case they send an **ACCEPT** message for the last learned value before sending the new proposal.

We say that a node *introduces a new value* w during the operation if it is the first node to send a $\langle \textbf{PROPOSE}, w \rangle$ for w in the interval of the operation. A node can introduce a new value coming from an internal source, i.e., the value was buffered and proposed when the node had already finished its previous proposal, or from an external source, i.e., after receiving a proposal originated from another node before the operation started.

Let v be the value received from the application call for *op* and e_C (as well as all previous events) be assigned round 0. If there are no active faulty nodes, a correct node learns a value containing v by at most round 7 (by Lemma 31, here, we include the time v can remain buffered). Also by the end of round 5, every correct node has sent a **PROPOSE** message for v and has v validated by the end of round 6 (including buffering time, a correct node proposes v in round 4 at the latest). By that point, all correct nodes are waiting for their proposals to complete and, therefore, cannot introduce a value from an internal source. In order to delay a correct node from leaning a value containing v by round 7, every correct node should receive a new value in a **PROPOSE** message before, which is added to *Pending* but is not validated. Using a simple inductive argument, $2k+1$ new proposals originated from faulty nodes are necessary to delay a correct node from learning a value from round 7 to round $7+2k$.

Suppose that there is an execution where it takes $8+2k+1$ rounds for node i to complete an operation. But there are only k active faulty nodes, which means that at least $k+1$ extra proposals were introduced by active faulty nodes.

Let f_0 be an active faulty node that introduced more than one of the $2k+1$ values that delayed the operation (assuming w.l.o.g. that there are *exactly* $2k+1$ new proposals). Let w and w' be the first and the second values introduced by f_0 respectively. If w' was received by f_0 from an internal source, f_0 should have finished its previous proposal (and learned a value containing w) before proposing w'. But because w was one of the values that delayed the operation, and since channels are FIFO, f_0 needs to add v to *Pending* before validating w (at least a majority of correct nodes sent a **PROPOSE** for v before sending a **PROPOSE** for w). f_0 then learns a value containing v and sends **ACCEPT** with that value to everyone. The **ACCEPT** message is received by correct processes before $\langle \textbf{PROPOSE}, w' \rangle$, and they would be able to adopt it.

So f_0 must have received $\langle \textbf{PROPOSE}, w' \rangle$ from an external source at most by round 1, which means it issued proposals for w' that can be received by at most round 2. We can also conclude that at least $k+1$ values were introduced by active faulty nodes from external sources. Now let w_{k+1} be the $(k+1)$th such value used to delay correct nodes from learning v. The earliest round w_{k+1} can delay is $7+k$, which means that by round $7+k$ all correct nodes already sent a propose for w_{k+1}, but by the end of round $5+k$ no correct node has done it (otherwise w_{k+1} would have been validated in round $7+k$ by every correct process). Take the first active faulty node f_1 from which a correct node received $\langle \textbf{PROPOSE}, w_{k+1} \rangle$. Since the earliest this message is received is in round $6+k$, the earliest it could be sent is

in round $5 + k$, so f_1 first received $\langle \mathbf{PROPOSE}, w_{k+1} \rangle$ from another distinct active faulty node, f_2, which sent it in round $4 + k$ the earliest. But w_{k+1} was introduced from an external source and it needs to be received by a faulty node at round 1. Following the chain above, for the node f_{k+6} to receive it in round 1, there would be necessary a chain of $k + 6$ active nodes, although there are only k.

Therefore, an operation takes less than $8 + 2k + 1$ rounds to complete. ◀

Perpetual Exploration in Anonymous Synchronous Networks with a Byzantine Black Hole

Adri Bhattacharya
Indian Institute of Technology Guwahati, Assam, India

Pritam Goswami
Sister Nivedita University, Kolkata, India

Evangelos Bampas
Université Paris-Saclay, CNRS, Laboratoire Interdisciplinaire des Sciences du Numérique, 91400 Orsay, France

Partha Sarathi Mandal
Indian Institute of Technology Guwahati, Assam, India

Abstract

In this paper, we investigate the following question:

"How can a group of initially co-located mobile agents perpetually explore an unknown graph, when one stationary node occasionally behaves maliciously, under the control of an adversary?"

This malicious node is termed as "Byzantine black hole (BBH)" and at any given round it may choose to destroy all visiting agents, or none of them. While investigating this question, we found out that this subtle power turns out to drastically undermine even basic exploration strategies which have been proposed in the context of a classical, always active, black hole.

We study this perpetual exploration problem in the presence of at most one BBH, without initial knowledge of the network size. Since the underlying graph may be 1-connected, perpetual exploration of the entire graph may be infeasible. Accordingly, we define two variants of the problem, termed as PerpExploration-BBH and PerpExploration-BBH-Home. In the former, the agents are tasked to perform perpetual exploration of at least one component, obtained after the exclusion of the BBH. In the latter, the agents are tasked to perform perpetual exploration of the component which contains the *home* node, where agents are initially co-located. Naturally, PerpExploration-BBH-Home is a special case of PerpExploration-BBH. The mobile agents are controlled by a synchronous scheduler, and they communicate via *face-to-face* model of communication.

The main objective in this paper is to determine the minimum number of agents necessary and sufficient to solve these problems. We first consider the problems in acyclic networks, and we obtain optimal algorithms that solve PerpExploration-BBH with 4 agents, and PerpExploration-BBH-Home with 6 agents in trees. The lower bounds hold even in path graphs. In general graphs, we give a non-trivial lower bound of $2\Delta - 1$ agents for PerpExploration-BBH, and an upper bound of $3\Delta + 3$ agents for PerpExploration-BBH-Home. To the best of our knowledge, this is the first paper that studies a variant of a black hole in arbitrary networks, without initial topological knowledge about the network.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Design and analysis of algorithms

Keywords and phrases mobile agents, perpetual exploration, malicious host, Byzantine black hole

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.16

Related Version *Full Version*: https://arxiv.org/abs/2508.07703

Funding *Adri Bhattacharya*: Supported by CSIR, Govt. of India, Grant Number: 09/731(0178)/2020-EMR-I.

Acknowledgements We thank the DISC 2025 reviewers for their careful reading and useful feedback.



© Adri Bhattacharya, Pritam Goswami, Evangelos Bampas, and Partha Sarathi Mandal;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

In distributed mobile agent algorithms, a fundamental task is the collaborative exploration of a network by a collection of mobile agents. It was introduced and formulated by Shannon [20] in 1951. Later, the real life applicability of this problem in fields like unmanned search and rescue, monitoring, network search, etc. has garnered a lot of interest from researchers across the world which leads them to study this problem in many different settings. Ensuring the security of agents against threats or breaches is one of the major concerns in the design of exploration algorithms. Among the various security threats, two types have received the most attention in the literature of mobile agent algorithms so far. These are threats from *malicious agents* [6, 14] and *malicious hosts* [4, 8, 9]. In this work, we are interested in the latter. A malicious host in a network is a stationary node that can destroy any incoming agent without leaving any trace. Dobrev et al. introduced this type of malicious node in their 2006 paper [9], referring to such malicious hosts as *black holes*. In this paper, we will use interchangeably the terms "classical black hole" and "black hole".

There is extensive literature on *Black Hole Search* (BHS) problem, that requires locating the black hole by multiple mobile agents in a network. The BHS problem has been studied under many different scenarios and under many different communication models ([3, 4, 5, 7, 9, 10, 11]). In all of these above mentioned works, the black hole is considered to be the classical black hole which always destroys any incoming agent without fail. In this work we are interested in a more general version of the classical black hole, called *Byzantine Black Hole*, or BBH [13]. A BBH has the choice to act, at any given moment, as a black hole, destroying any data present on that node, or as a non-malicious node. Moreover, it is assumed that the initial position of the agents is safe. Note that the BHS problem does not have an exact equivalent under the Byzantine black hole assumption. Indeed, if the BBH always acts as an non-malicious node, then it can never be detected by any algorithm. Thus, in this work, our goal is not to locate the Byzantine black hole (BBH), but rather to perpetually explore the network despite its presence. Note that, if all agents are initially co-located and the BBH is a cut-vertex of the network, then it becomes impossible to visit every vertex, as the BBH can block access by behaving as a black hole. Consequently, in this work, we modify the exploration problem to focus on perpetually exploring *one connected component* of the graph, among the connected components which would be obtained if the BBH and all its incident edges was removed from the network. In [13], the problem was first introduced as PerpExploration-BBH for a ring network, where it was studied under various communication models and also under any initial deployment. In this work, in addition to investigating PerpExploration-BBH, we focus on a variant in which agents are required to perpetually explore specifically the connected component that includes their initial position, referred to as the *home*. We call this variant PerpExploration-BBH-Home. PerpExploration-BBH-Home is particularly relevant in practical scenarios where the starting vertex serves as a central base, for example to aggregate the information collected by individual agents or as a charging station for energy-constrained agents. Therefore, during perpetual exploration, it is essential to ensure that the component being explored by the agents includes their *home*. Note that the two variants are equivalent if the BBH is not a cut vertex.

1.1 Related work

Exploration of underlying topology by mobile agents in presence of a malicious host (also termed as black hole) has been well studied in literature. This problem, also known as black hole search or BHS problem was first introduced by Dobrev et al. [9]. Detailed surveys of

the problems related to black holes can be found in [16, 17, 19]. Královič and Miklík [15, 18] were the first to introduce some variants of the classical black hole, among which the *gray hole*, a Byzantine version of the black hole (controlled by the adversary) with the ability, in each round, to behave either as a regular node or as a black hole. They considered the model, in which each agents are controlled by an asynchronous scheduler, where the underlying network is a ring with each node containing a whiteboard (i.e., each node can store some data, with which the agents can communicate among themselves). In this model, one of the problems they solved is Periodic Data Retrieval in presence of a gray hole with 9 agents. Later, Bampas et al. [1] improved the results significantly under the same model. For the gray hole case, they obtained an optimal periodic data retrieval strategy with 4 agents. It may be noted that BBH behavior coincides with the "gray hole" considered in these papers. Moreover, periodic data retrieval is similar to perpetual exploration, as the main source of difficulty lies in periodically visiting all non-malicious nodes in order to collect the generated data. Goswami et al. [13] first studied the perpetual exploration problem in presence of BBH (i.e., PerpExploration-BBH) in ring networks, where the agents are controlled by a synchronous scheduler. They studied this problem under different communication models (i.e., *face-to-face*, *pebble* and *whiteboard*, where *face-to-face* is the weakest and *whiteboard* is the strongest model of communication), for different initial positions of the agents, i.e., for the case when the agents are initially co-located and for the case when they are initially scattered. Specifically for the case where the agents communicate *face-to-face* and they are initially co-located at a node, they obtained an upper bound of 5 agents and a lower bound of 3 agents for PerpExploration-BBH.

1.2 Our contributions

We study the PerpExploration-BBH and PerpExploration-BBH-Home problems in specific topologies and in arbitrary synchronous connected networks with at most one BBH under the *face-to-face* communication model (i.e., two agents can only communicate if both of them are on the same node). To the best of our knowledge, all previous papers in the literature which study variants of the classical black hole, do so assuming that the underlying network is a ring. Our primary optimization objective is the number of agents required to solve these problems.

For tree networks, we provide a tight bound on the number of agents for both problems. Specifically, we show that 4 agents are necessary and sufficient for PerpExploration-BBH in trees, and that 6 agents are necessary and sufficient for PerpExploration-BBH-Home in trees. Our algorithms work without initial knowledge of the size n of the network. However, knowledge of n would not reduce the number of agents, as our lower bounds do not assume that n is unknown. Note that our 4-agent PerpExploration-BBH algorithm can be used to directly improve the 5-agent algorithm for rings, with knowledge of the network size, given in [13] in the same model (*face-to-face* communication, initially co-located agents).

To simplify the presentation, we present these results for path networks, and then we explain how to adapt the algorithms to work in trees with the same number of agents.

In the case of general network topologies, we propose an algorithm that solves the problem PerpExploration-BBH-Home, hence also PerpExploration-BBH, with $3\Delta + 3$ agents, where Δ is the maximum degree of the graph, without knowledge of the size of the graph.

In terms of lower bounds, we first show that, if the BBH behaves as a classical black hole (i.e., if it is activated in every round), then at least $\Delta + 1$ agents are necessary for perpetual exploration, even with knowledge of n. In the underlying graph used in this proof, the BBH is not a cut vertex, hence the lower bound holds even for PerpExploration-BBH. This

can be seen as an analogue, in our model, of the well-known $\Delta + 1$ lower bound for black hole search in asynchronous networks [8]. In passing, under the same assumption of the BBH behaving as a classical black hole, we discuss an algorithm that performs perpetual exploration with only $\Delta + 2$ agents, without knowledge of n.

We then use the full power of the Byzantine black hole to prove a stronger, and more technical, lower bound of $2\Delta - 1$ agents. In this last lower bound, the structure and, indeed, the size of the graph is decided dynamically based on the actions of the algorithm. Hence, the fact that agents do not have initial knowledge of n is crucial in this proof. In this case, the BBH may be a cut vertex, but the adversarial strategy that we define in the proof never allows any agents to visit any other component than the one containing the home node. Therefore, this lower bound also carries over to PerpExploration-BBH.

The above results are the first bounds to be obtained for a more powerful variant of a black hole, for general graphs, under the assumption that the agents have no initial topological knowledge about the network.

Several technical details and proofs are omitted due to lack of space, and they can be found in the full version of the paper.

2 Model and basic definitions

The agents operate in a simple, undirected, connected port-labeled graph $G = (V, E, \lambda)$, where $\lambda = \big(\lambda_v\big)_{v \in V}$ is a collection of port-labeling functions $\lambda_v : E_v \to \{1, \ldots, \delta_v\}$, where E_v is the set of edges incident to node v and δ_v is the degree of v. We denote by n the number of nodes and by Δ the maximum degree of G.

An algorithm is modeled as a deterministic Turing machine. Agents are modeled as instances of an algorithm (i.e., copies of the corresponding deterministic Turing machine) which move in G. Each agent is initially provided with a unique identifier.

The execution of the system proceeds in synchronous rounds. In each round, each agent receives as input the degree of its current node, the local port number through which it arrived at its current node, i.e. $\lambda_v(\{u, v\})$ if it just arrived at node v by traversing edge $\{u, v\}$, or 0 if it did not move in the last round, and the configurations of all agents present at its current node. It then computes the local port label of the edge that it wishes to traverse next (or 0 if it does not wish to move). All agents are activated, compute their next move, and perform their moves in simultaneous steps within a round. We assume that all local computations take the same amount of time and that edge traversals are instantaneous.

Note that we will only consider initial configurations in which all agents are co-located on a node called "the *home*". In this setting, the set of unique agent identifiers becomes common knowledge in the very first round.

At most one of the nodes $\mathfrak{b} \in V$ is a Byzantine black hole. In each round, the adversary may choose to activate the black hole. If the black hole is activated, then it destroys all agents that started the round at $\mathfrak{b}$, as well as all agents that choose to move to $\mathfrak{b}$ in that round. The agents have no information on the position of the Byzantine black hole, except that it is not located at the *home* node. Furthermore, the agents do not have initial knowledge of the size of the graph.

2.1 Problem definition

We define the Perpetual Exploration problem with initially co-located agents in the presence of a Byzantine black hole, hereafter denoted PerpExploration-BBH, as the problem of perpetually exploring at least one of the connected components resulting from the removal of the Byzantine black hole from the graph. If the graph does not contain a Byzantine black hole, then the entire graph must be perpetually explored.

If, in particular, the perpetually explored component *must* be the component containing the *home*, then the corresponding problem is denoted as PerpExploration-BBH-Home.

▶ **Definition 1.** *An* instance *of the PerpExploration-BBH problem is a tuple* $\langle G, k, h, \mathfrak{b}\rangle$, *where* $G = (V, E, \lambda)$ *is a connected port-labeled graph,* $k \geq 1$ *is the number of agents starting on the home* $h \in V$, *and* $\mathfrak{b} \in (V \setminus \{h\}) \cup \{\perp\}$ *is the node that contains the Byzantine black hole. If* $\mathfrak{b} = \perp$, *then* G *does not contain a Byzantine black hole.*

For the following definitions, fix a PerpExploration-BBH instance $I = \langle G, k, h, \mathfrak{b}\rangle$, where $G = (V, E, \lambda)$, and let $\mathcal{A}$ be an algorithm.

▶ **Definition 2.** *We say that an execution of* $\mathcal{A}$ *on* I perpetually explores a subgraph H of G *if every node of* H *is visited by some agent infinitely often.*

▶ **Definition 3.** *Let* $C_1, C_2, \ldots, C_t$ *be the connected components of the graph* $G - \mathfrak{b}$, *resulting from the removal of* $\mathfrak{b}$ *and all its incident edges from* G. *If* $\mathfrak{b} = \perp$, *then* $t = 1$ *and* $C_1 \equiv G$. *Without loss of generality, let* $h \in C_1$.

We say that $\mathcal{A}$ solves PerpExploration-BBH on I, *if for every execution starting from the initial configuration in which* k *agents are co-located at node* h, *at least one of the components* $C_1, C_2, \ldots, C_t$ *is perpetually explored.*

We say that $\mathcal{A}$ solves PerpExploration-BBH-Home on I, *if for every execution starting from the initial configuration in which* k *agents are co-located at node* h, *the component* C_1 *(containing the home) is perpetually explored.*

Finally, we say that $\mathcal{A}$ *solves PerpExploration-BBH with* k_0 *agents* if it solves the problem on any instance with $k \geq k_0$ agents (similarly for PerpExploration-BBH-Home). Note that any algorithm that solves PerpExploration-BBH-Home also solves PerpExploration-BBH.

3 Perpetual exploration in path and tree networks

In this section, our main aim is to establish the following two theorems, giving the optimal number of agents that solve PerpExploration-BBH and PerpExploration-BBH-Home in path graphs.

▶ **Theorem 4.** *4 agents are necessary and sufficient to solve PerpExploration-BBH in path graphs, without initial knowledge of the size of the graph.*

▶ **Theorem 5.** *6 agents are necessary and sufficient to solve PerpExploration-BBH-Home in path graphs, without initial knowledge of the size of the graph.*

For the necessity part, we prove that there exists no algorithm solving PerpExploration-BBH (resp. PerpExploration-BBH-Home) with 3 agents (resp. 5 agents), even assuming knowledge of the size of the graph. For the sufficiency part of Theorem 5, we provide an algorithm, which we call Path_PerpExplore-BBH-Home, solving PerpExploration-BBH-Home with 6 initially co-located agents, even when the size of the path is unknown to the agents. Thereafter, we modify this algorithm to solve PerpExploration-BBH with 4 initially co-located agents, thus establishing the sufficiency part of Theorem 4.

In Section 3.1, we give a brief sketch of the approach we have used to prove the lower bounds on the number of agents. Then, in Section 3.2, we describe the algorithms.

3.1 Lower bounds in paths

▶ **Theorem 6.** *At least 4 agents are necessary to solve* PerpExploration-BBH *in all path graphs with $n \geq 9$ nodes, even assuming initial knowledge of the size of the graph.*

In order to prove Theorem 6, we establish, in a series of lemmas, several different types of configurations from which the adversary can force an algorithm to fail, by destroying all agents or by failing to entirely perpetually explore any component. Omitting a lot of details, these configurations are as follows:

- If one agent remains alive, and at least two BBH locations are consistent with the history of the agent up to that time, then the algorithm fails.
- If two agents remain alive on the same side *of the path* of at least three potential BBH locations, then the algorithm fails. Also, if two agents remain alive on either side of at least three potential BBH locations, then the algorithm fails.
- If three agents remain alive on the same side of at least eight potential BBH locations, of which at least the first three are consecutive nodes in the path, then the algorithm fails.

Theorem 6 then follows by observing that, in the initial configuration with three co-located agents, all nodes apart from the home node are potential BBH positions.

▶ **Theorem 7.** *At least 6 agents are necessary to solve* PerpExploration-BBH-Home *in all path graphs with $n \geq 145$ nodes, even assuming initial knowledge of the size of the graph.*

The first element of the proof of Theorem 7 is that at least one agent must remain at the *home* node up to at least the destruction of the first agent by the BBH (actually, up to the time of destruction of the first agent plus the time it would take for that information to arrive at the home node), otherwise the adversary can trap all agents in the wrong component, i.e., the one not containing the *home* node.

Then, very informally, we argue that, no matter how the algorithm divides the 5 available agents into a group of *exploring* agents that must go arbitrarily far from the home node, and a group of *waiting* agents that must wait at the home node, either the number of *exploring* agents is not enough for them to explore arbitrarily far from the *home* node and, at the same time, be able to detect exactly the BBH position if it is activated, or the number of *waiting* agents is not enough to allow them to recover the information of the exact BBH position from one of the remaining *exploring* agents that has been stranded on the other side of the BBH.

3.2 Description of Algorithm Path_PerpExplore-BBH-Home

Here we first describe the algorithm, assuming that the BBH does not intervene and destroy any agent. Then we describe how the agents behave after the BBH destroys at least one agent (i.e., the BBH intervenes).

We call this algorithm Path_PerpExplore-BBH-Home. Let $\langle P, 6, h, \mathfrak{b} \rangle$ be an instance of PerpExploration-BBH-Home, where $P = (V, E, \lambda)$ is a port-labeled path. Per Definition 3, all agents are initially co-located at h (the home node). To simplify the presentation, we assume that h is an extremity of the path. Our algorithm works with 6 agents. Initially, among them the four least ID agents will start exploring P, while the other two agents will wait at h, for the return of the other agents. We first describe the movement of the four least ID agents say, a_0, a_1, a_2 and a_3, on P. Based on their movement, they identify their role as follows: a_0 as F, a_1 as I_2, a_2 as I_1 and a_3 as L. The identities F, I_2, I_1 and L are denoted as Follower, Intermediate2, Intermediate1 and Leader, respectively. The exploration is performed by these four agents in two steps, in the first step, they form a

particular pattern on P. Then in the second step, they move collaboratively in such a way that the pattern is translated from the previous node to the next node in five rounds. Since the agents do not have the knowledge of n, where $|V| = n$, they do the exploration of P, in $\log n$ phases, and then this repeats. In the i-th phase, the four agents start exploring P by continuously translating the pattern to the next node, starting from h, and moves up to a distance of 2^i from h. Next, it starts exploring backwards in a similar manner, until they reach h. It may be observed that, any phase after $j-$th phase (where $2^j \geq n$), the agents behave in a similar manner, as they behaved in $j-$th phase, i.e., they move up to 2^j distance from h, and then start exploring backwards in a similar manner, until each agent reaches h. Note that, this can be done with agents having $\mathcal{O}(\log n)$ bits of memory, in order to store the current phase number.

We now describe how the set of four agents, i.e., L, I_1, I_2 and F create and translate the pattern to the next node.

Creating pattern. L, I_1, I_2 and F takes part in this step from the very first round of any phase, starting from h. In the first round, L, I_1 and I_2 moves to the next node. Then in the next round only I_2 returns back to *home* to meet with F. Note that, in this configuration the agents L, I_1, I_2 and F are at two adjacent nodes while F and I_2 are together and L and I_1 are together on the same node. We call this particular configuration the *pattern*. We name the exact procedure as Make_Pattern.

Translating pattern. After the pattern is formed in first two rounds of a phase, the agents translate the pattern to the next node until the agent L reaches either at one end of the path graph P, or, reaches a node at a distance 2^i from h in the i-th phase. Let, v_0, v_1, v_2 be three consecutive nodes on P, where, suppose L and I_1 is on v_1 and F and I_2 is on v_0. This translation of the pattern makes sure that after 5 consecutive rounds L and I_1 is on v_2 and F and I_2 is on v_1, thus translating the pattern by one node. We call these 5 consecutive rounds where the pattern translates starting from a set of 2 adjacent nodes, say v_0, v_1, to the next two adjacent nodes, say v_1, v_2, a *sub-phase* in the current phase. The description of the 5 consecutive rounds i.e., a sub-phase without the intervention of the BBH at $\mathfrak{b}$ (such that $\mathfrak{b} \in \{v_0, v_1, v_2\}$) are as follows.

Round 1: L moves to v_2 from v_1.
Round 2: I_2 moves to v_1 from v_0 and L moves to v_1 back from v_2.
Round 3: I_2 moves back to v_0 from v_1 to meet with F. Also, L moves back to v_2 from v_1.
Round 4: F and I_2 moves to v_1 from v_0 together.
Round 5: I_1 moves to v_2 from v_1 to meet with L.

So after completion of round 5 the pattern is translated from nodes v_0, v_1 to v_1, v_2. The pictorial description of the translate pattern is explained in Fig. 1.

Now, suppose v_2 is the node upto which the pattern was supposed to translate at the current phase (or, it can be the end of the path graph also). So, when L visits v_2 for the first time, in round 1 of some sub-phase it knows that it has reached the end of the path graph for the current phase. Then in the same sub-phase at round 2 it conveys this information to I_2 and I_1. In round 3 of the same sub-phase, F gets that information from I_2. So at the end of the current sub-phase all agents has the information that they have explored either one end of the path graph or the node upto which they were supposed to explore in the current phase. In this case, they interchange the roles as follows: the agent which was previously had role L changes role to F, the agent having role F changes it to L, I_1 changes role to I_2 and I_2 changes role to I_1 (refer to Fig. 2). Then from the next sub-phase onwards they

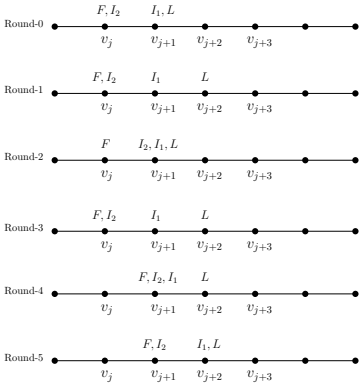


Figure 1 Depicts translating pattern steps.

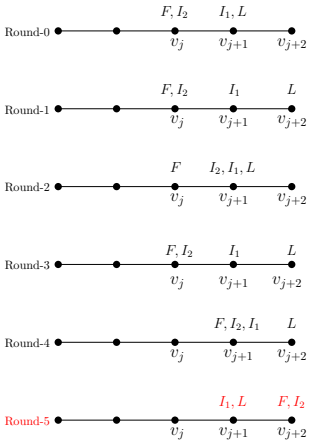


Figure 2 Depicts the step-wise interchange of roles, when the agents reach the end of the path graph, while performing translate pattern.

start translating the pattern towards h. It may be noted that, once L (previously F) reaches h in round 1 of a sub-phase, it conveys this information to the remaining agents in similar manner as described above. So, at round 5 of this current sub-phase, F (previously L) and I_2 (previously I_1) also reaches h, and meets with L (previously F) and I_1 (previously I_2). We name the exact procedure as TRANSLATE_PATTERN.

Intervention by the BBH. Now we describe the behaviour of the agents while the BBH kills at least one exploring agent during create pattern or, during translate pattern. We first claim that, however the BBH intervenes while the four agents are executing creating pattern or translating pattern, there must exists at least one agent, say A_{alive} which knows the exact location of the BBH. The justification of the claim is immediate if we do case by case studies fixing a BBH position and the round it is intervening with the agents. This can be during creating pattern or in a particular sub-phase of translating pattern. Now, there can be two cases:

Case-I: Let $A_{alive} \in C_1$ (C_1 being the connected component of $G - \mathfrak{b}$ that contains h) then it can perpetually explore every vertex in C_1 satisfying the requirement.

Case-II: Let $A_{alive} \in C_2$ (C_2 being the connected component of $G - \mathfrak{b}$ such that $h \notin C_2$). In this case A_{alive} places itself to the adjacent node of the BBH on C_2. Let T_i be the maximum time, required for the 4 agents (i.e., L, I_1, I_2 and F) to return back to h in $i-$th phase if BBH does not intervene. Let us denote the aing agents at h, i.e., a_4, a_5 as F_1, F_2. Starting from the i-th phase, they wait for T_i rounds for the other agents to return. Now, if the set of agents L, I_1, I_2 and F fail to return back to *home* within T_i rounds in phase, i, the agents F_1 and F_2 starts moving *cautiously*. In the cautious move, first F_1 visits the next node and in the next round, returns back to the previous node to meet with F_2, that was waiting there for F_1. If F_1 fails to return then F_2 knows the exact location of $\mathfrak{b}$, which is the next node F_1 visited. In this case, F_2 remains in the component of h, hence can explore it perpetually. Otherwise, if F_1 returns back to F_2, then in the next round both of them moves together to the next node.

It may be noted that, A_{alive} knows which phase is currently going on and so it knows the exact round at which F_1 and F_2 starts moving cautiously. Also, it knows the exact round at which F_1 first visits $\mathfrak{b}$, say at round r. A_{alive} waits till round $r-1$, and at round r it

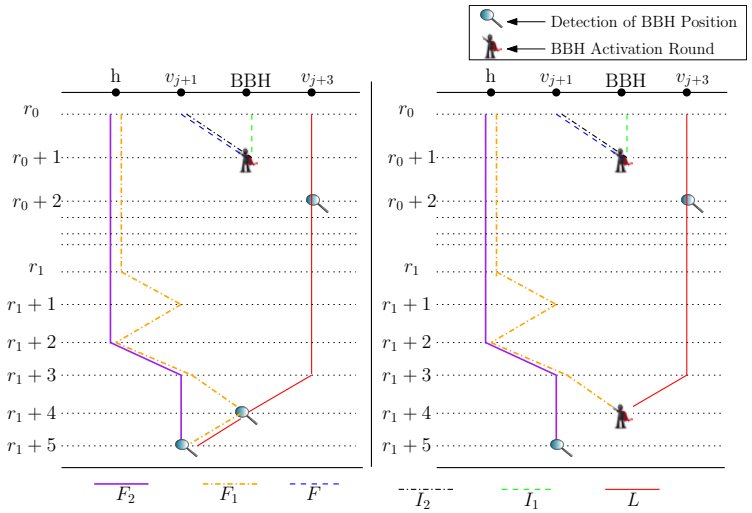


Figure 3 Represents the time diagram, in which at least one among F_1 and F_2 detects $\mathfrak{b}$, and perpetually explores the path graph.

moves to $\mathfrak{b}$. Now at round r, if adversary activates $\mathfrak{b}$, it destroys both F_1 and A_{alive} then, F_1 fails to return back to F_2 in the next round. This way, F_2 knows the exact location of $\mathfrak{b}$, while it remains in C_1. So it can explore C_1 by itself perpetually. The right figure in Fig. 3, represents the case where L detects $\mathfrak{b}$ at round r_0+2, and waits till round r_1+3. In the meantime, at round r_1 (where $r_1 = r'_0 + T_i$, $r'_0 < r_0$ and r'_0 is the first round of phase i), F_1 and F_2 starts moving cautiously. Notably, along this movement, at round r_1+4, F_1 visits $\mathfrak{b}$, and at the same time L as well visits $\mathfrak{b}$ from v_{j+3}. The adversary activates $\mathfrak{b}$, and both gets destroyed. So, at round r_1+5, F_2 finds failure of F_1's return and understands the next node to be $\mathfrak{b}$, while it is present in C_1. Accordingly, it perpetually explores C_1.

On the other hand, if at round r, adversary doesn't activate $\mathfrak{b}$, then F_1 meets with A_{alive} and knows that they are located on the inactivated $\mathfrak{b}$. In this case they move back to C_1 and starts exploring the component C_1, avoiding $\mathfrak{b}$. The left figure in Fig. 3 explores this case, where L detects the position of $\mathfrak{b}$ at round r_0+2, and stays at v_{j+3} until r_1+3, then at round r_1+4, when F_1 is also scheduled to visit $\mathfrak{b}$, L also decides to visit $\mathfrak{b}$. But, in this situation, the adversary does not activate $\mathfrak{b}$ at round r_1+4, so both F_1 and L meets, gets the knowledge from L that they are on $\mathfrak{b}$. In the next round, they move to v_{j+1} which is a node in C_1, where they meet F_2 and shares this information. After which, they perpetually explore C_1.

The correctness of the algorithm follows from the description of the algorithm and the case by case study of all possible interventions by the BBH. We have thus established the sufficiency part of Theorem 5.

▶ Note 8. It may be noted that, our algorithm Path_PerpExplore-BBH-Home, without F_1 and F_2, is sufficient to solve PerpExploration-BBH. It is because, if A_{alive} in C_i ($i \in \{1,2\}$) after one intervention by the BBH, it can perpetually explore C_i without any help from F_1 and F_2 as it knows the exact location of the BBH. This proves the sufficiency part of Theorem 4.

▶ Remark 9. Note that our 4-agent PerpExploration-BBH algorithm for paths directly improves the 5-agent algorithm for rings with knowledge of the network size, given in [13], in the *face-to-face* model with initially co-located agents. Indeed, the 4-agent pattern can

keep moving around the ring as if in a path graph. Naturally, it will never reach a node of degree 1. If and when the BBH destroys an agent, then, as we showed, at least one agent remains alive and knows the position of the BBH. As the agent knows the size of the ring in this case, it can perform perpetual exploration of the non-malicious ring nodes, without ever visiting the BBH.

3.2.1 Modification of the path algorithms to work in trees

We modify the algorithms for path graphs of Section 3.2 to work for trees, with the same number of agents. The algorithms for trees work by translating the pattern from one node to another by following the $k-$`Increasing-DFS` [12] algorithm up to a certain number of nodes in a certain phase.

4 Perpetual exploration in general graphs

In this section, we establish upper and lower bounds on the optimal number of agents required to solve Perpetual Exploration in arbitrary graphs with a BBH, without any initial knowledge about the graph. In particular, we give a lower bound of $2\Delta-1$ agents for PerpExploration-BBH (Theorem 10 below), which carries over directly to PerpExploration-BBH-Home, and an algorithm for PerpExploration-BBH-Home using $3\Delta+3$ agents (Theorem 12 below), which also solves PerpExploration-BBH.

We give a brief sketch of the lower bound proof in Section 4.1, and we present the algorithm in Section 4.2.

4.1 Lower bound in general graphs

For the lower bound, we construct a particular class of graphs in which any algorithm using $2\Delta-2$ agents or less must fail.

▶ **Theorem 10.** *For every $\Delta \geq 4$, there exists a class of graphs $\mathcal{G}$ with maximum degree Δ, such that any algorithm using at most $2\Delta-2$ agents, with no initial knowledge about the graph, fails to solve PerpExploration-BBH in at least one of the graphs in $\mathcal{G}$.*

Proof sketch. For a fixed $\Delta \geq 4$, we construct the corresponding graph class $\mathcal{G}$ by taking an underlying path $\mathcal{P}$, consisting of two types of vertices $\{v_i : 1 \leq i \leq \Delta\}$ and, for each $i \leq \Delta$, the nodes $\{u_j^i : 1 \leq j \leq l_i\}$, where $l_i \geq 1$. The nodes $\{u_j^i\}$ form a subpath of length l_i+1 connecting v_i to v_{i+1}. The nodes v_i (for $\in \{1,2,\ldots,\Delta-1\}$) are special along $\mathcal{P}$, because they are connected to the BBH $\mathfrak{b}$ either directly or via a new node w_i. Every node of $\mathcal{P}$, as well as $\mathfrak{b}$, now completes its degree up to Δ by connecting to at most $\Delta-1$ trees of height 2 (see example in Figure 4). Every node w_i completes its degree up to Δ by connecting to $\Delta-2$ new nodes.

Note that, to reach $\mathfrak{b}$ from a vertex of the form u_j^i, it is required to visit either v_i or v_{i+1}. In the example of Figure 4, we have $\Delta=4$, $l_1=l_2=2$ and $l_3=1$, so in $\mathcal{P}$ there are two vertices u_1^1, u_2^1 between v_1 and v_2, two vertices u_1^2, u_2^2 between v_2 and v_3, and one vertex u_1^3 between v_3 and v_4. In addition, v_2 is directly connected to $\mathfrak{b}$ (or BBH) whereas v_1 and v_3 are connected to $\mathfrak{b}$ via a path of length 2.

Given an algorithm $\mathcal{A}$ which claims to solve PerpExploration-BBH with $2\Delta-2$ co-located agents, the adversary returns a port-labeled graph $G=(V,E,\lambda) \in \mathcal{G}$ in which $\mathcal{A}$ fails, by choosing the lengths l_i and the connection between v_i and $\mathfrak{b}$ (direct or through w_i). A high-level idea of the proof is as follows. First, it can be ensured that, $\mathcal{A}$ must instruct at

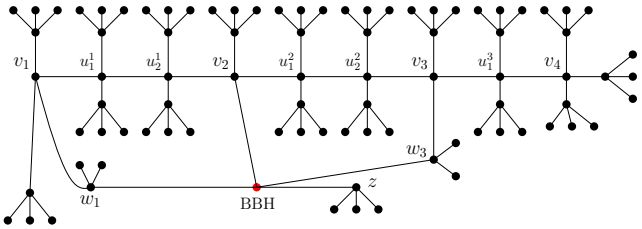


Figure 4 An example graph of the class $\mathcal{G}$, used in the proof of Theorem 10.

least one agent to visit a node which is at 2 hop distance from v_i, for all $i \in \{1, 2, \ldots, \Delta - 1\}$. Let the agent which first visits a node which is at 2 hop distance from v_i, starts from v_i at time t_i. **Case-1**: Now, based on $\mathcal{A}$, if at t_i, at least 2 agents are instructed to move from v_i along the same port, then the adversary chooses a graph from $\mathcal{G}$, such that v_i is connected to $\mathfrak{b}$, and returns a port labeling, such that at $t_i + 1$, these agents reach $\mathfrak{b}$. **Case-2**: Otherwise, the adversary chooses a graph from $\mathcal{G}$ such that, either v_i is connected to $\mathfrak{b}$ or there exists an intermediate vertex w_i connecting v_i with $\mathfrak{b}$. In this case also, the adversary returns the port labeling such that, the agent must move to $\mathfrak{b}$ (or w_i) at $t_i + 1$ and to some vertex z (or $\mathfrak{b}$) at t_i' (where $t_i' > t_i + 1$).

In addition, the graph chosen by the adversary sets the distance between v_{i-1} and v_i in a way to ensure that, within the time interval $[t_i, t_i']$, no agent from any v_α ($\alpha \in \{0, 1, \ldots, i-1\}$) attempts to visit $\mathfrak{b}$. So, for **Case-1**, two agents directly gets destroyed. For **Case-2**, after the first agent is destroyed, remaining agents do not understand which among the next 2 nodes from v_i is $\mathfrak{b}$. Now, for $\mathcal{A}$ to succeed, at least one other agent from v_i gets destroyed, at some round t_i''. This shows that from each v_i, for all $i \in \{1, 2, \ldots, \Delta - 1\}$ at least 2 agents each, are destroyed. Thus, $\mathcal{A}$ fails. ◀

4.2 Description of Algorithm Graph_PerpExplore-BBH-Home

Here we discuss the algorithm, termed as Graph_PerpExplore-BBH-Home that solves PerpExploration-BBH-Home on a general graph, $G = (V, E, \lambda)$. We will show that our algorithm requires at most $3\Delta + 3$ agents. Let $\langle G, 3\Delta + 3, h, \mathfrak{b} \rangle$ be an instance of the problem PerpExploration-BBH-Home, where $G = (V, E, \lambda)$ is a simple port-labeled graph. The structure of our algorithm depends upon four separate algorithms Translate_Pattern along with Make_Pattern (discussed in Section 3.2), Explore (explained in this section) and BFS-Tree-Construction [2]. So, before going in to details of our algorithm that solves PerpExploration-BBH-Home, we recall the idea of BFS-Tree-Construction.

An agent starts from a node $h \in V$ (also termed as *home*), where among all nodes in G, only h is marked. The agent performs breadth-first search (BFS) traversal, while constructing a BFS tree rooted at h. The agent maintains a set of edge-labeled paths, $\mathcal{P} = \{P_v \ : \ \text{edge labeled shortest path from } h \text{ to } v, \forall v \in V \text{ such that the agent has visited } v\}$ while executing the algorithm. During its traversal, whenever the agent visits a node w from a node u, then to check whether the node w already belongs to the current BFS tree of G constructed yet, it traverses each stored edge labeled paths in the set $\mathcal{P}$ from w one after the other, to find if one among them takes it to the marked node h. If yes, then it adds to its map a cross-edge (u, w). Otherwise, it adds to the already constructed BFS tree, the node w, accordingly $\mathcal{P} = \mathcal{P} \cup P_w$ is updated. The underlying data structure of Root_Paths [2] is used to perform these processes. This strategy guarantees as per Proposition 9 of [2], that BFS-Tree-Construction algorithm constructs a map of G, in presence of a marked node, within $\mathcal{O}(n^3\Delta)$ steps and using $\mathcal{O}(n\Delta \log n)$ memory, where $|V| = n$ and Δ is the maximum degree in G.

In our algorithm, we use k agents (in Theorem 12, it is shown that $k = 3\Delta + 3$ agents are sufficient), where they are initially co-located at a node $h \in V$, which is termed as *home*. Initially, at the start our algorithm asks the agents to divide in to three groups, namely, `Marker`, `SG` and $\texttt{LG}_0$, where `SG` (or smaller group) contains the least four ID agents, the highest ID agent among all k agents, denoted as `Marker` stays at h (hence h acts as a marked node), and the remaining $k - 5$ agents are denoted as $\texttt{LG}_0$ (or larger group). During the execution of our algorithm, if at least one member of $\texttt{LG}_0$ detects one port leading to the BBH from one of its neighbor, in that case at least one member of $\texttt{LG}_0$ settles down at that node, acting as an *anchor* blocking that port which leads to the BBH, and then some of the remaining members of $\texttt{LG}_0$ forms $\texttt{LG}_1$. In general, if at least one member of $\texttt{LG}_i$ detects the port leading to the BBH from one of its neighbors, then again at least one member settles down at that node acting as an *anchor* to block that port leading to the BBH, and some of the remaining members of $\texttt{LG}_i$ forms $\texttt{LG}_{i+1}$, such that $|\texttt{LG}_{i+1}| < |\texttt{LG}_i|$. It may be noted that, a member of $\texttt{LG}_i$ only settles at a node v (say) acting as an *anchor*, only if no other *anchor* is already present at v. Also, only if a member of $\texttt{LG}_i$ settles as an *anchor*, then only some of the members of $\texttt{LG}_i$ forms $\texttt{LG}_{i+1}$.

In addition to the groups $\texttt{LG}_0$ and `SG`, the `Marker` agent permanently remains at h. In a high-level the goal of our Graph_PerpExplore-BBH-Home algorithm is to create a situation, where eventually at least one agent blocks, each port of C_1 that leads to the BBH (where $C_1, C_2, \ldots, C_t$ are the connected components of $G - \mathfrak{b}$, such that $h \in C_1$), we term these blocking agents as *anchors*, whereas the remaining alive agents must perpetually explore at least C_1.

Initially from h, the members of `SG` start their movement, and the members of $\texttt{LG}_0$ stays at h until they find that, none of the members of `SG` reach h after a certain number of rounds. Next, we explain one after the other how both these groups move in G.

Movement of `SG`. The members (or agents) in `SG` works in phases, where in each phase the movement of these agents are based on the algorithms Make_Pattern and Translate_Pattern (both of these algorithms are described in Section 3.2). Irrespective of which, the node that they choose to visit during *making pattern* or *translating pattern* is based on the underlying algorithm BFS-Tree-Construction.

More specifically, the i-th phase (for some $i > 0$) is divided in two sub-phases: i_1-th phase and i_2-th phase. In the i_1-th phase, the members of `SG` makes at most 2^i translations, while executing the underlying algorithm BFS-Tree-Construction. Next, in the i_2-th phase, irrespective of their position after the end of i_1-th phase, they start translating back to reach h. After they reach h during the i_2-th phase, they start $(i+1)$-th phase (which has again, $(i+1)_1$ and $(i+1)_2$ sub-phase). Note that, while executing i_1-th phase, if the members of `SG` reach h, in that case they continue executing i_1-th phase. We already know as per Section 3.2, each translation using Translate_Pattern requires 5 rounds and for creating the pattern using Make_Pattern it requires 2 rounds. This concludes that, it requires at most $Ti_j = 5 \cdot 2^i + 2$ rounds to complete i_j-th phase, for each $i > 0$ and $j \in \{1, 2\}$.

If at any point, along their traversal, the adversary activates the BBH, such that it interrupts the movement of `SG`. In that scenario, at least one member of `SG` must remain alive, exactly knowing the position of the BBH from its current node (refer to the discussion of **Intervention by the BBH** in Section 3.2). The agent (or agents) which knows the exact location of the BBH, stays at the node adjacent to the BBH, such that from its current node, it knows the exact port that leads to the BBH, or in other words they act as *anchors* with respect to one port, leading to the BBH. In particular, let us suppose, the agent holds the adjacent node of BBH, with respect to port α from BBH, then this agent is termed as `Anchor(`α`)`.

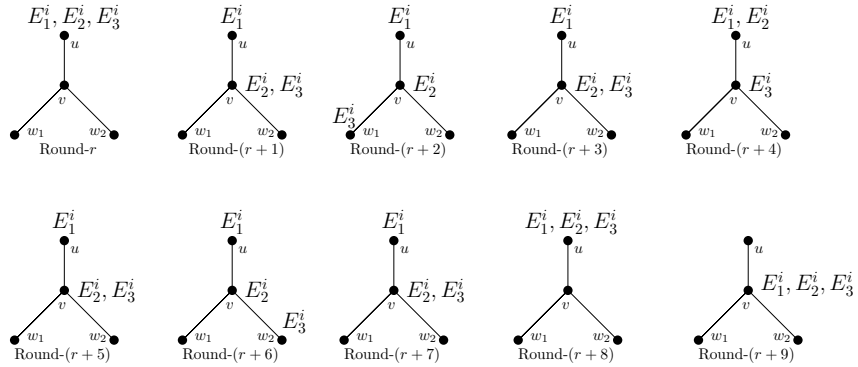


Figure 5 Depicts the round-wise execution of Explore(v) from u by the explorer agents of $\texttt{LG}_i$, for some $i \geq 0$ on the neighbors w_1 and w_2 of v.

Movement of $\texttt{LG}_0$. These group members stay at h with `Marker`, until the members of `SG` are returning back to h in the i_2-th phase, for each $i > 0$. If all members of `SG` do not reach h, in the i_2-th phase, i.e., within Ti_2 rounds since the start of i_2-th phase, then the members of $\texttt{LG}_0$ start their movement.

Starting from h, the underlying movement of the members of $\texttt{LG}_0$ is similar to BFS-Tree-Construction, but while moving from one node to another they do not execute neither Make_Pattern nor Translate_Pattern, unlike the members of `SG`. In this case, if all members of $\texttt{LG}_0$ are currently at a node $u \in V$, then three lowest ID members of $\texttt{LG}_0$ become the explorers, they are termed as E_1^0, E_2^0 and E_3^0 in increasing order of their IDs, respectively. If based on the BFS-Tree-Construction, the next neighbor to be visited by the members of $\texttt{LG}_0$ is v, where $v \in N(u)$, then the following procedure is performed by the explorers of $\texttt{LG}_0$, before $\texttt{LG}_0$ finally decides to visit v.

Suppose at round r (for some $r > 0$), $\texttt{LG}_0$ members reach u, then at round $r+1$ both E_2^0 and E_3^0 members reach v. Next at round $r+2$, E_3^0 traverses to the first neighbor of v and returns to v at round $r+3$. At round $r+4$, E_2^0 travels to u from v and meets E_1^0 and then at round $r+5$ it returns back to v. This process iterates for each neighbor of v, and finally after each neighbor of v is visited by E_3^0, at round $r + 4 \cdot (\delta_v - 1) + 1$ both E_2^0 and E_3^0 returns back to u. And in the subsequent round each members of $\texttt{LG}_0$ visit v. The whole process performed by E_1^0, E_2^0 and E_3^0 from u is termed as Explore(v), where v symbolizes the node at which the members of $\texttt{LG}_0$ choose to visit from a neighbor node u. After the completion of Explore(v), each member of $\texttt{LG}_0$ (including the explorers) visit v from u. A pictorial description is explained in Fig. 5.

It may be noted that, if the members of $\texttt{LG}_0$ at the node u, according to the BFS-Tree-Construction algorithm, are slated to visit the neighboring node v, then before executing Explore(v), the members of $\texttt{LG}_0$ checks if there exists an *anchor* agent blocking that port which leads to v. If that is the case, then $\texttt{LG}_0$ avoids visiting v from u, and chooses the next neighbor, if such a neighbor exists and no *anchor* agent is blocking that edge. If no such neighbor exists from u to be chosen by $\texttt{LG}_0$ members, then they backtrack to the parent node of u, and start iterating the same process.

From the above discussion we can have the following remark.

▶ Remark 11. If at some round t, the explorer agents of $\texttt{LG}_i$ (i.e., E_1^i, E_2^i and E_3^i), are exploring a two length path, say $P = u \to v \to w$, from u, then all members of $\texttt{LG}_i$ agrees on P at t. This is due to the fact that the agents while executing Explore(v) from u must follow the path $u \to v$ first. Now from v, E_3^i chooses the next port in a particular pre-decided

order (excluding the port through which it entered v). So, whenever it returns back to v to meet E_2^i after visiting a node w, E_2^i knows which port it last visited and which port it will chose next and relay that information back to other agents of $\texttt{LG}_i$ on u. So, after E_2^i returns back to v again from u when E_3^i starts visiting the next port, all other agents know about it.

During the execution of Explore(v) from u, the agent E_3^0 can face one of the following situations:

- It can find an *anchor* agent at v, acting as $\texttt{Anchor}(\beta)$, for some $\beta \in \{1, \ldots, \delta_v\}$. In that case, during its current execution of Explore(v), E_3^0 does not visit the neighbor of v with respect to the port β.
- It can find an *anchor* agent at a neighbor w (say) of v, acting as $\texttt{Anchor}(\beta')$, where $\beta' \in \{1, \ldots, \delta_w\}$. If the port connecting w to v is also β', then E_3^0 understands v is the BBH, and accordingly tries to return to u, along the path $w \to v \to u$, and if it is able to reach to u, then it acts as an *anchor* at u, with respect to the edge (u, v). On the other hand, if port connecting w to v is not β', then E_3^0 continues its execution of Explore(v).

The agent E_2^0 during the execution of Explore(v), can encounter one of the following situations, and accordingly we discuss the consequences that arise due to the situations encountered.

- It can find an *anchor* agent at v where the *anchor* agent is not E_3^0, in which case it continues to execute Explore(v).
- It can find an *anchor* agent at v and finds the *anchor* agent to be E_3^0. In this case, E_2^0 returns back to u, where $\texttt{LG}_1$ is formed, where $\texttt{LG}_1 = \texttt{LG}_0 \setminus \{E_3^0\}$. Next, the members of $\texttt{LG}_1$ start executing the same algorithm from u, with new explorers as E_1^1, E_2^1 and E_3^1.
- E_2^0 can find that E_3^0 fails to return to v from a node w (say), where $w \in N(v)$. In this case, E_2^0 understands w to be the BBH, and it visits u in the next round to inform this to remaining members of $\texttt{LG}_0$ in the next round, and then returns back to v, and becomes $\texttt{Anchor}(\beta)$, where $\beta \in \{1, \ldots, \delta_v\}$ and β is the port for (v, w). On the other hand, $\texttt{LG}_0$ after receiving this information from E_2^0, transforms to $\texttt{LG}_1$ (where $\texttt{LG}_1 = \texttt{LG}_0 \setminus \{E_2^0, E_3^0\}$) and starts executing the same algorithm, with E_1^1, E_2^1 and E_3^1 as new explorers.

Lastly, during the execution of Explore(v) the agent E_1^0 can face the following situation.

- E_2^0 fails to return from v, in this situation E_1^0 becomes $\texttt{Anchor}(\beta)$ at u, where β is the port connecting u to v. Moreover, the remaining members of $\texttt{LG}_0$, i.e., $\texttt{LG}_0 \setminus \{E_1^0, E_2^0, E_3^0\}$ forms $\texttt{LG}_1$ and they start executing the same algorithm from u, with new explorers, namely, E_1^1, E_2^1 and E_3^1, respectively.

For each E_1^0, E_2^0 and E_3^0, if they do not face any of the situations discussed above, then they continue to execute Explore(v).

A pictorial explanation of two scenarios, of how an *anchor* settles at neighbor nodes of BBH is explained in Fig. 6.

Correctness. In order to prove the correctness of the algorithm Graph_PerpExplore-BBH-Home, we give a brief sketch of the proof of the following theorem.

▶ **Theorem 12.** *Algorithm Graph_PerpExplore-BBH-Home solves PerpExploration-BBH-Home in arbitrary graphs with $3\Delta + 3$ agents, having $\mathcal{O}(n\Delta \log n)$ memory, without initial knowledge about the graph.*

Proof sketch. We first show that if $\mathfrak{b}$ (i.e., BBH) never destroys any agent then the problem PerpExploration-BBH-Home is solved only by the four agents in $\texttt{SG}$ (follows from the correctness of BFS-Tree-Construction). On the otherhand if BBH destroys any

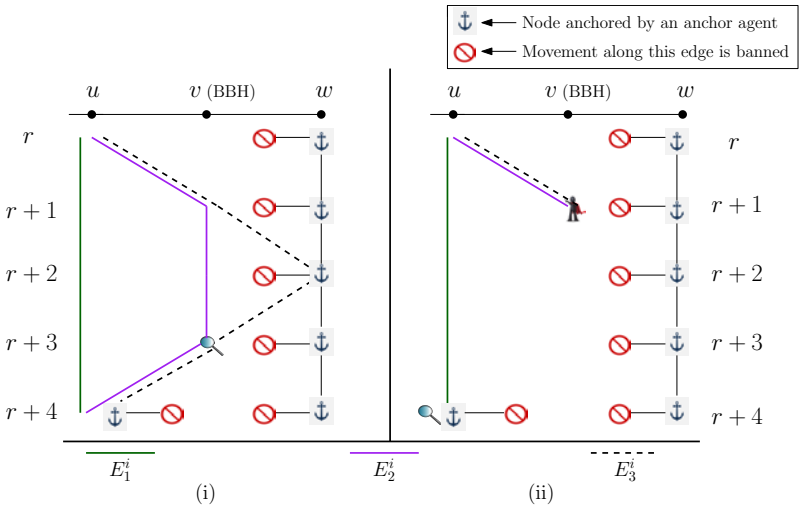


Figure 6 Depicts the time diagram of EXPLORE(v) of $\texttt{LG}_i$ along a specific path $P = u \to v \to w$, where $v = \mathfrak{b}$, in which w contains an *anchor* agent for the edge (v, w). In (i), it depicts even if $\mathfrak{b}$ is not activated, an explorer agent settles as an *anchor* at u for the edge (u, v). In (ii), it depicts that activation of $\mathfrak{b}$ destroys both E_2^i and E_3^i, then an explorer E_1^i gets settled as *anchor* at u for (u, v).

agent, then we prove that there exists an agent from $\texttt{SG}$ acting as an *anchor* at a node in $N(\mathfrak{b})$, where $N(\mathfrak{b})$ indicates the neighbors of $\mathfrak{b}$. The node can be either in C_1 or C_j, where $G - \mathfrak{b} = C_1 \cup \cdots \cup C_t$ for some $t \geq 1$ and *home* $\in C_1$ and $j \neq 1$. Now if we assume, BBH destroys at least one agent then, we define a set $\mathcal{U}$ of nodes in G, where $\mathcal{U} \subseteq N(\mathfrak{b}) \cap C_1$ such that no vertex in $\mathcal{U}$ contains an *anchor*. A node $u \in \mathcal{U}$ ceases to exist in $\mathcal{U}$, whenever an agent visits u and blocks the edge $(u, \mathfrak{b})$, by becoming an *anchor*. We show that during the execution of our algorithm, $\mathcal{U}$ eventually becomes empty. In addition to that, we also show that, the agents in $\texttt{LG}_i$ (for some $i \geq 0$) never visit a node which is not in C_1. By $\texttt{LG}_i$ visiting a node, we mean to say that all agents are located at that node simultaneously. This concludes that, all neighbors in C_1 of $\mathfrak{b}$, gets anchored by an *anchor* agent. So, eventually all the agents, which are neither an *anchor* nor a `Marker`, can perpetually explore C_1. Also, to set up an *anchor* at each neighbor of $\mathfrak{b}$, at most 2 agents are destroyed. So, this shows that $3(\Delta - 1)$ agents are required to *anchor* each neighbor of C_1 by the members of $\texttt{LG}_i$ (for any $i \geq 0$). So, in total $3\Delta + 3$ agents are sufficient to execute GRAPH_PERPEXPLORE-BBH-HOME, including `Marker`, 4 agents in $\texttt{SG}_0$, and one agent which is neither a `Marker` nor an *anchor*, i.e., the one which perpetually explores C_1. Moreover, to execute BFS-TREE-CONSTRUCTION as stated in [2, Proposition 9], each agent requires $\mathcal{O}(n\Delta \log n)$ memory. This concludes the proof. ◀

4.3 Perpetual exploration in presence of a black hole

In the special case in which the Byzantine black hole is activated in each round, i.e., behaves as a classical black hole, we show that the optimal number of agents for perpetual exploration (we call this problem PERPEXPLORATION-BH) drops drastically to between $\Delta + 1$ and $\Delta + 2$ agents.

The lower bound holds even with initial knowledge of n, and even if we assume that agents have knowledge of the structure of the graph, minus the position of the black hole and the local port labelings at nodes. This can be seen as an analogue, in our model, of the well-known $\Delta + 1$ lower bound for black hole search in asynchronous networks [8].

Details of the proof and a discussion of the algorithm that solves the problem with $\Delta + 2$ agents can be found in the full version.

5 Conclusion

We gave the first non-trivial upper and lower bounds on the optimal number of agents for perpetual exploration, in presence of at most one Byzantine black hole in general, unknown graphs.

One noteworthy point, as regards the related problem of *pinpointing* the location of the malicious node, is that all our algorithms have the property that, even in an execution in which the Byzantine black hole destroys only one agent, all remaining agents manage to determine exactly the position of the malicious node, at least within the component which ends up being perpetually explored. By contrast, this is not the case for the algorithms proposed in [13] for PerpExploration-BBH in ring networks, as the adversary can time a single activation of the Byzantine black hole, destroying at least one agent, so that the remaining agents manage to perpetually explore the ring, but without ever being able to disambiguate which one of two candidate nodes is the actual malicious node.

A few natural open problems remain. First, close the important gap between $2\Delta - 1$ and $3\Delta + 3$ for the optimal number of agents required for PerpExploration-BBH and PerpExploration-BBH-Home, in general graphs. Second, note that our general graph lower bound of $2\Delta - 1$ holds only for graphs of maximum degree at least 4 (Theorem 10). Could there be a 4-agent algorithm for graphs of maximum degree 3? Third, in the special case of a black hole (or, equivalently, if we assume that the Byzantine black hole is activated in each round), close the gap between $\Delta + 1$ and $\Delta + 2$ agents. Fourth, our 4-agent algorithm in paths induced a direct improvement of the optimal number of agents for perpetual exploration in a ring with known n, under the face-to-face communication model, from 5 agents (which was shown in [13]) to 4 agents. However, it is still unknown whether the optimal number of agents is 3 or 4 in this case.

Finally, it would be interesting to consider stronger or weaker variants of the Byzantine black hole, and explore the tradeoffs between the power of the malicious node and the optimal number of agents for perpetual exploration. Orthogonally to this question, one can consider different agent communication models or an asynchronous scheduler.

References

1 Evangelos Bampas, Nikos Leonardos, Euripides Markou, Aris Pagourtzis, and Matoula Petrolia. Improved periodic data retrieval in asynchronous rings with a faulty host. *Theor. Comput. Sci.*, 608:231–254, 2015. doi:10.1016/J.TCS.2015.09.019.

2 Jérémie Chalopin, Shantanu Das, and Adrian Kosowski. Constructing a map of an anonymous graph: Applications of universal sequences. In *Principles of Distributed Systems: 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings 14*, pages 119–134. Springer, 2010. doi:10.1007/978-3-642-17653-1_10.

3 Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Complexity of searching for a black hole. *Fundamenta Informaticae*, 71(2-3):229–242, 2006. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-05.

4 Jurek Czyzowicz, Dariusz Kowalski, Euripides Markou, and Andrzej Pelc. Searching for a black hole in synchronous tree networks. *Combinatorics, Probability and Computing*, 16(4):595–619, 2007. doi:10.1017/S0963548306008133.

5 Giuseppe Antonio Di Luna, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Black hole search in dynamic rings. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 987–997. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00098.

6 Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Trans. Algorithms*, 11(1):1:1–1:28, 2014. doi:10.1145/2629656.

7 Stefan Dobrev, Paola Flocchini, Rastislav Královič, and Nicola Santoro. Exploring an unknown dangerous graph using tokens. *Theoretical Computer Science*, 472:28–45, 2013. doi:10.1016/J.TCS.2012.11.022.

8 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Comput.*, 19(1):1–35, 2006. doi:10.1007/S00446-006-0154-Y.

9 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48:67–90, 2007. doi:10.1007/S00453-006-1232-Z.

10 Stefan Dobrev, Nicola Santoro, and Wei Shi. Locating a black hole in an un-oriented ring using tokens: The case of scattered agents. In *European Conference on Parallel Processing*, pages 608–617. Springer, 2007. doi:10.1007/978-3-540-74466-5_64.

11 Paola Flocchini, David Ilcinkas, and Nicola Santoro. Ping pong in dangerous graphs: Optimal black hole search with pebbles. *Algorithmica*, 62:1006–1033, 2012. doi:10.1007/S00453-011-9496-3.

12 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005. doi:10.1016/J.TCS.2005.07.014.

13 Pritam Goswami, Adri Bhattacharya, Raja Das, and Partha Sarathi Mandal. Perpetual exploration of a ring in presence of byzantine black hole. In Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni, editors, *28th International Conference on Principles of Distributed Systems, OPODIS 2024, December 11-13, 2024, Lucca, Italy*, volume 324 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.OPODIS.2024.17.

14 Wayne A Jansen. Intrusion detection with mobile agents. *Computer Communications*, 25(15):1392–1401, 2002. doi:10.1016/S0140-3664(02)00040-3.

15 Rastislav Královic and Stanislav Miklík. Periodic data retrieval problem in rings containing a malicious host. In Boaz Patt-Shamir and Tínaz Ekim, editors, *Structural Information and Communication Complexity, 17th International Colloquium, SIROCCO 2010, Sirince, Turkey, June 7-11, 2010. Proceedings*, volume 6058 of *Lecture Notes in Computer Science*, pages 157–167. Springer, 2010. doi:10.1007/978-3-642-13284-1_13.

16 Euripides Markou. Identifying hostile nodes in networks using mobile agents. *Bull. EATCS*, 108:93–129, 2012. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/52.

17 Euripides Markou and Wei Shi. Dangerous graphs. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 455–515. Springer, 2019. doi:10.1007/978-3-030-11072-7_18.

18 Stanislav Miklík. *Exploration in faulty networks*. PhD thesis, Comenius University, Bratislava, 2010.

19 Mengfei Peng, Wei Shi, Jean-Pierre Corriveau, Richard Pazzi, and Yang Wang. Black hole search in computer networks: State-of-the-art, challenges and future directions. *Journal of Parallel and Distributed Computing*, 88:1–15, 2016. doi:10.1016/J.JPDC.2015.10.006.

20 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.

Deterministic Synchronous Self-Stabilizing BFS Construction with Constant Space Complexity

Lélia Blin
Université Paris Cité, CNRS
IRIF, F-75013 Paris, France

Franck Petit
Sorbonne Université, CNRS
LIP6, Paris, France

Sébastien Tixeuil
Sorbonne Université, CNRS
LIP6, IUF, Paris, France

Abstract

In this paper, we resolve a long-standing open problem in self-stabilization asking whether it is possible to construct a spanning tree using *constant* memory per node in a synchronous semi-uniform networks, i.e., networks in which one node is distinguished. We design a synchronous self-stabilizing algorithm that constructs a breadth-first search (BFS) tree in any *anonymous* semi-uniform network using only a constant number of bits of memory per node. Crucially, our approach operates without any prior knowledge of global network parameters such as maximum degree, diameter, or number of nodes. In contrast to traditional self-stabilizing methods – such as pointer-to-neighbors, distance-to-root, or identifiers – that are unsuitable under strict memory constraints, our solution employs an innovative constant-space token dissemination mechanism. This mechanism effectively eliminates cycles and rectifies errors in the BFS structure, ensuring both correctness and memory efficiency. The proposed algorithm not only meets the stringent requirements of memory-constrained distributed systems, but also opens new avenues for research in the design of self-stabilizing protocols under severe resource limitations: constant space-complexity may not systematically prevent the existence of self-stabilizing algorithms for important non-trivial tasks.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics

Keywords and phrases Distributed algorithms, fault-tolerance, transient faults, self-stabilization, memory optimization

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.17

Funding *Lélia Blin*: Additional support from ANR projects ENEDISC (ANR-24-CE48-7768-01), and from the InIDEX Project METALG.
Franck Petit: Additional support from ANR project ANR SKYDATA (ANR-22-CE25-0008-02).
Sébastien Tixeuil: Additional support from ANR project SAPPORO (2019-CE25-0005).

1 Introduction

This paper addresses the challenge of developing memory-efficient self-stabilizing algorithms for the Breadth-First Search (BFS) Spanning Tree Construction problem. Self-stabilization [3, 18, 20, 23] is a versatile technique that facilitates recovery after arbitrary *transient* faults impact the distributed system, affecting both the participating processes and the communication medium. Essentially, a self-stabilizing protocol can restore the system to a legitimate configuration (from which its behavior satisfies its specification), beginning from an arbitrary, potentially corrupted, initial global state, without the need of any human intervention. A BFS spanning tree construction involves *(i)* each node identifying a single


© Lélia Blin, Franck Petit, and Sébastien Tixeuil;
licensed under Creative Commons License CC-BY 4.0



neighbor as its parent, with the exception of the root node, which has no parent, and *(ii)* the set of parent-child relationships forming an acyclic connected structure, ensuring that following the parent nodes from any node results in the shortest path to the root node. Memory efficiency pertains to the amount of information transmitted to neighboring nodes to enable stabilization. A smaller space complexity results in reduced information transmission, which *(i)* decreases the overhead of self-stabilization in the absence of faults or after stabilization, and *(ii)* facilitates the integration of self-stabilization and replication [22, 24].

We adopt the classical *state model* introduced in [18]. In this model, a node can read its own memory and the memory of its neighbors, and execute its algorithm accordingly when needed. This means that a node's memory is accessible to all its neighbors, ensuring they all obtain the same information. Consequently, if a node v wants to communicate information exclusively to its neighbor u, v must indicate in its memory that the information is intended only for u. This indication is typically made by adding node u's identifier. This type of communication is inherent to wireless networks, where nodes exchange information by broadcasting messages over one or more carrier waves. In such networks, the notion of *n*eighboring node of v refers to any other node able to directly receiving messages emitted by v, typically without the aid of relays. Consequently, targeting a specific neighbor necessitates that the sender explicitly includes the recipient's ID within the transmitted message. This identification requirement adds a significant overhead in memory space, specifically $O(\log n)$ bits in a network containing n nodes. Within anonymous semi-uniform networks, nodes are devoid of identifiers, and only a single node assumes a role distinct from the rest. More precisely, the distinguished node operates under a dedicated protocol, whereas all other nodes execute an identical protocol. As a result of anonymity, it is not possible to rely on identifiers to interact with or refer to a specific node.

In [21], the authors introduce the notion of a *silent algorithm* (defined in a model slightly different from the state model [18]), a fundamental concept that significantly influences the space complexity of self-stabilizing algorithms. An algorithm is said to be *silent* if every execution eventually reaches a point beyond which the state of all nodes remains unchanged. They show that in networks of n nodes, solving tasks such as leader election or spanning tree construction silently requires at least $\Omega(\log n)$ bits of memory per node. However, in the state model, a self-stabilizing algorithm is not necessary silent, and *talkative* (*i.e.*, non-silent) approaches were successful in reducing the memory cost of self-stabilizing algorithms [10, 11].

In general networks, such as those considered in this paper, self-stabilizing leader election is closely linked to self-stabilizing tree construction. On one hand, the presence of a leader enables efficient self-stabilizing tree construction [9, 13, 15, 16, 19, 25, 26, 27]. On the other hand, growing and merging trees is the primary technique for designing self-stabilizing leader election algorithms in networks, as the leader often serves as the root of an inward tree [1, 2, 4, 7, 10, 11].

From a memory perspective, the aforementioned works employ two algorithmic techniques for self-stabilization that negatively impact memory complexity. The first technique involves using a *distance* variable to store the distance of each node to the elected node in the network. This distance variable is employed in self-stabilizing spanning tree construction to break cycles resulting from arbitrary initial states, see for instance [1, 2, 4]. Clearly, storing distances in n-node networks may require $\Omega(\log n)$ bits per node. Some works [6, 10, 11] distribute pieces of information about the distances to the leader among the nodes using different mechanisms, allowing for the storage of $o(\log n)$ bits per node. However, in general networks, the best result so far [11] uses $O(\log\log n + \log\Delta)$ bits per node, closely matched by the $\Omega(\log\log n)$ bits per node lower bound obtained in [8].

The second technique involves using a *pointer-to-neighbor* variable to unambiguously designate a specific neighbor for each node (with the designated neighbor being aware of this). For tree construction, pointer-to-neighbor variables typically store the parent node in the constructed tree (with the parent being aware of its children). In a naive implementation, the parent of each node is designated by its identifier, requiring $\Omega(\log n)$ bits for each pointer variable. However, it is possible to reduce the memory requirement to $O(\log \Delta)$ bits per pointer variable in networks with a maximum degree of Δ by using node-coloring at distance 2 instead of identifiers to identify neighbors. The best available deterministic self-stabilizing distance-2 coloring protocol [11] achieves a memory footprint of $O(\log \log n + \log \Delta)$ bits per node, assuming unique identifiers. This implies that even self-stabilizing protocols solely relying on a constant number of pointer-to-neighbor variables per node [26, 16] actually use at best $O(\log \log n + \log \Delta)$ bits per node (assuming unique identifiers) in the classical state model, and that a lower bound of $\Omega(\log \Delta)$ bits per node also holds for these solutions (regardless of the nodes being anonymous or not).

We now turn to the results of Itkis and Levin [25], which constitute one of the works most closely related to ours. their approach, the authors propose a randomized asynchronous self-stabilizing algorithm for leader election, based on a *deterministic* construction of a BFS tree. Their algorithms are designed under the same model as in this paper, namely the *state model*, but augmented with *pointer-to-neighbor variables*. As already mentioned above, using such additional assumption, each node u can maintain a unique pointer to a neighbor $v \in N(u)$; more importantly, this neighbor v is aware that it is being pointed to by u. Three deterministic BFS construction algorithms are presented by Itkis and Levin [25], proving successive improvements on their space complexity. Their first algorithm is a standard BFS construction based on the distance from each node to the root, and it is requiring $O(\log n)$ bits of memory per node. The other two algorithms are presented in natural language, and their formalization as a set of guarded rules is left undefined. The former is restricted to graphs of degree 2. It is based on a sophisticated and elegant cycle-detection mechanism based on the Thue-Morse sequence. Note that although no formal proof exist in the literature, the inherent properties of the Thue-Morse sequence and finite automata theory suggest that it is impossible to implement this sequence within finite memory. Nevertheless, the authors use a Thue-Morse-based certificate to obtain a constant space complexity per node. The second of the two algorithms generalizes the first to arbitrary graphs. This algorithm maintains a forest of pointers, and guarantees acyclicity of these pointers thanks to the use of the degree-2 algorithm. The latter is applied to each path from the root of the BFS tree to its leaves, where each path is identified thanks to a depth-first search (DFS) traversal of the BFS tree. These DFS traversals are benefiting from the pointer-to-neighbors variables. To keep the space complexity constant, the DFS traversals are performed successively. This requires to keep track of which DFS traversals have already been performed. This authors argue that this can be done using constant space complexity using the pointer-to-neighbor variables allowed by their model. However, in absence of such pointers, it is not clear how to implement this sequence of DFS traversals using less than $\Omega(\log \Delta)$ bits of memory per node [15, 17].

Overall, up to this paper, there exists no deterministic self-stabilizing solution to the spanning tree construction problem that uses a constant number of bits of memory per process. Furthermore, the reuse of known techniques makes this goal unattainable.

Our contribution. We introduce the first deterministic self-stabilizing Breath-First Search Spanning Tree construction algorithm that utilizes a constant amount of memory per node in the classical state model, irrespective of network parameters such as degree, diameter, or size. Our algorithm functions in arbitrary topology networks and does not depend on any global knowledge.

In [8], it is shown that constructing a self-stabilizing spanning tree without a pre-existing leader using constant memory is impossible. Therefore, a necessary assumption for achieving constant space per node is that the anonymous network is semi-uniform, with a single node designated as the leader, serving as the root of the spanning tree, while all other nodes are anonymous and uniformly execute the same algorithm. Additionally, for synchronization purposes, our solution assumes a fully synchronous network, where all nodes operate at the same pace in a lock-step manner.

Our solution relies on a few key components.

First, each non-root node maintains a *rank* variable, which classifies neighbors as parents (if their rank is lower), siblings (if their rank is the same), or children (if their rank is higher). In the stabilized phase, the rank value corresponds to the distance to the root modulo three (hence, the BFS property of the constructed spanning tree), an idea already suggested in various non self-stabilizing settings [5, 12, 14]. The self-stabilizing property adds a significant challenge to the task, as the initial configuration may make the rank variable induce cycles or multiple roots, from which the algorithm must recover. Since identifying the single parent of each node (if it exists) cannot make use of node identifiers (as all non-root nodes are anonymous), instead we select the parent node (among the set of parents) whose port number is minimal. However, nodes do *not* explicitly communicate the specific parent they have selected, as this would necessitate $O(\log \Delta)$ bits of memory.

The second component of our algorithm uses a similar technique as the *Power Supply* introduced by Afek and Bremler [1]. In both cases, the root continuously floods the network with tokens that have a limited lifespan to circumvent one of the main issues associated with token-based approaches: the infinite circulation of tokens within a cycle. The token's lifespan is controlled by a binary counter, where each bit is stored on a distinct node. So, the tokens generated by the root decay exponentially (half of the tokens disappear at the root's neighbors, half of the remaining ones at the neighbors' downward neighbors, and so on). Tokens are used to destroy cycles and correct inconsistent situations that may appear in arbitrary initial configurations, observing that in synchronous networks, tokens passed deterministically arrive simultaneously at all nodes equidistant from the root (and thus at all neighbors with the same rank for any given node, if the ranks are correct). The exponential decay prevents tokens from looping indefinitely in the network, without relying on any global knowledge.

Overall, our algorithm uses only 6 bits of memory per node (actually, only 36 states), and has a stabilization time of $O(2^{\varepsilon})$ time units, where ε denotes the root eccentricity (unknown to the participating nodes).

2 Preliminaries

In this section, we define the underlying distributed execution model considered in this paper, and state what it means for a protocol to be self-stabilizing.

A *distributed system* is an undirected connected graph, $G = (V, E)$, where V is a set of vertices (or nodes) – $|V| = n,\ n \geq 2$ – and E is the set of edges connecting those vertices. The distance between two nodes u and v, denoted by d_v^u is the length of the shortest path

between u and v in G. Note that since G is not oriented, $d_v^u = d_u^v$, and if $u = v$, then $d_v^u = 0$. Then, the eccentricity of a node v is the maximum distance from v to any other node in the network.

Vertices represent processes, and edges represent bidirectional communication links (that is, the ability for two nodes to communicate directly). We assume that each node v is able to distinguish each incident edge with a locally assigned unique label called *port number*. Port numbers are immutable and are not coordinated in any way: the edge (u, v) may correspond to port number k at u and to port number $k' \neq k$ at v. Let $N(v)$ be the set of port numbers, also called *set of neighbors* of v. The *degree* of v denotes $|N(v)|$.

We consider the state model in the context of semi-uniform anonymous networks. In this model, the communication paradigm allows each node to read its own memory as well as the memories of its neighbors. Let v be any node in V, and let $\{u_0, \ldots, u_{|N(v)|}\}$ denote its set of neighbors. No neighbor $u_i,\ i \in \{0, \ldots, |N(v)|\}$, can determine which port number in $N(v)$ corresponds to the edge (v, u_i). Conversely, since nodes have no identifiers, v cannot designate any particular u_i to receive v's information. Hence, a pointer-to-neighbor cannot be implemented solely by reading the neighbors' variables.

A semi-uniform anonymous network is characterized by the absence of node identifiers and the presence of a unique distinguished node. This node serves as the root of the BFS tree and follows a distinct protocol, while all other nodes adhere to a uniform protocol. The protocol of a node consists of a set of variables and a set of (guarded) rules, of the following form:

$$< label >: < guard > \longrightarrow < statement >$$

Each node may write its own variables, and read its own variables and those of its neighboring nodes. The guard of a rule is a predicate on the variables of v and its neighbors. The statement of a rule of v updates one or more variables of v. A rule may be executed only if its guard evaluates to true. The rules are atomically executed, so the evaluation of a guard and the execution of the corresponding statement, if any, are done in one atomic step.

The *state* of a node is defined by the values of its variables. The *configuration* of a system is the product of the states of all nodes. Let Γ be the set of all possible configurations of the system. A distributed algorithm (or, protocol) $\mathcal{A}$ is a collection of binary transition relations, denoted by $\mapsto$, on Γ such that given two configurations γ_1 and γ_2, $\gamma_1 \mapsto \gamma_2$ by the atomic execution of guarded action of one or more nodes. A protocol $\mathcal{A}$ induces an oriented graph $\Gamma^{\mapsto} = (\Gamma, \mapsto)$, called the *transition graph* of $\mathcal{A}$. A sequence $e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots$, $\forall i \geq 0, \gamma_i \in \Gamma$, is called an *execution* of $\mathcal{A}$ if and only if $\forall i \geq 0, \gamma_i \mapsto \gamma_{i+1}$.

A node v is said to be *enabled* in a configuration $\gamma \in \Gamma$ if there exists a rule R such that the guard of R is true at v in γ. In this paper, we assume that each transition of $\Gamma^{\mapsto}$ is driven by a *synchronous scheduler*. This means that, given a pair of configurations $\gamma_1, \gamma_2 \in \Gamma$, then $\gamma_1 \mapsto \gamma_2 \in \Gamma^{\mapsto}$ if and only if all enabled nodes in γ_1 execute an atomic rule during the transition $\gamma_1 \mapsto \gamma_2$.

Self-Stabilization. A predicate P is *closed* for a transition graph $\Gamma^{\mapsto}$ if and only if every configuration in any execution e starting from a configuration satisfying P also satisfies P. A predicate Q is an *attractor* of P, denoted $P \rhd Q$, if and only if: *(i)* Q is closed for $\Gamma^{\mapsto}$, and *(ii)* for every execution e on $\Gamma^{\mapsto}$ beginning in a configuration satisfying P, e contains at least one configuration satisfying Q. A transition graph $\Gamma^{\mapsto}$ is *self-stabilizing* for a predicate P when P is an attractor of the predicate *true* (formally, *true* $\rhd P$), ensuring convergence to P from any initial configuration.

Problem to be solved. We assume a *rooted graph*, meaning that the set of vertices includes a specific node called the *root*, denoted by r. Let us call V^r the set V with the extra property that each node $v \in V$ is labeled with d_v^r. For every node $v \neq r$, let $Pred_v$ be the set of neighboring nodes of v in V^r that are labeled with $d_v^r - 1$. Let E^r be the subset of edges ($E^r \subset E$) such that for every edge $uv \in E^r$, either u belongs to $Pred_v$, or v belongs to $Pred_u$. So, in $G^r = (V^r, E^r)$, no two neighbors have the same label. Then $\overrightarrow{G^r}$ is the directed version of G^r every edge is oriented toward the smaller label. By construction, $\overrightarrow{G^r}$ is a Directed Acyclic Graph (DAG) rooted in r such that every decreasing label path from every $v \in V^r$ is a path of minimum length between v and r. Note that this DAG is uniquely defined. In the remainder of the paper, we refer to $\overrightarrow{G^r}$ as $BFS\text{-}DAG$ (rooted at r).

Let $BFS^{\equiv 3}$ be the projection of the $BFS\text{-}DAG$, where each vertex v is labeled with $d_v^r \bmod 3$. The problem we consider in this paper consists in the design of a self-stabilizing distributed algorithm that builds the $BFS^{\equiv 3}$ using a constant amount of memory. From this construction, one can retrieve the BFS tree by choosing for each node (except the root) the parent port number of minimum value.

3 Algorithm

3.1 Overview of the algorithm

Our algorithm formally described in Algorithm 1 aims to build a distributed version of $BFS^{\equiv 3}$, meaning each node v computes its distance to the root r modulo 3. Our algorithm makes use of an infinite flow of tokens originating from the root downwards the BFS-DAG. Of course, since the initial configuration is arbitrary, the nodes' variables may not originally match the $BFS^{\equiv 3}$, so the tokens may not initially flow according to the BFS-DAG.

For this algorithm to function properly, it is crucial that only the root *creates* tokens. When the root creates a token, all of its neighbors take that token in the next synchronous step, which implies that after one step, there are as many tokens as neighbors of the root. This process is then repeated from parents to children (according to the distance modulo 3 variables).

Since our model is synchronous, all tokens generated by the root at a given step progress while preserving an identical distance from the root.

A node v updates its distance modulo 3 based on the distance (modulo 3) of the neighbors from which it receives one or more tokens at the same synchronous step; if multiple neighbors send tokens, v merges them into a single token.

A common issue when using tokens to construct a spanning tree starting from an arbitrary configuration is the possibility of infinite token circulation within cycles, which prevents a valid spanning tree from forming. This problem may occur if tokens (not generated by the root) are already present in the initial configuration.

To address this, our algorithm ensures that each token has a limited lifespan, thereby avoiding infinite circulation. This limited lifespan is based on the following idea: each node maintains a bit, whose value indicate whether the token should be forwarded downwards; each time a token is received, the bit is flipped, effectively dropping half of the tokens that flow through the node. Globally, this process can be seen as creating a binary word along the paths from the root to the nodes. Tokens move along these paths, effectively performing binary addition one by one. A node v is allowed to collect a token if and only if all upward nodes having sent their tokens have their bit set to 1 before the addition. Otherwise, node v does not forward the tokens, leading to their disappearance.

Consider an execution of this approach on a chain of length ℓ, where all nodes v_1 through v_ℓ have their bits set to 0. Consequently, the binary word of length ℓ is $0000\ldots00$, note that the root does not contribute to this binary word. For simplicity, assume that a token moves from one neighbor to the next in just one step. If the root offers a token, its neighbor v_0 does not pass the token to its neighbor v_1 but changes its bit to 1. The resulting binary word is $1000\ldots00$. Next, when v_1 acquires the token, it can pass the token to v_2. At this point, v_1 changes its bit to 0 and v_2 changes its bit to 1, but the token is not forwarded to v_3 and thus disappears. The resulting binary word is $0100\ldots00$. This process continues accordingly. From this scenario, observe that the token's lifespan is very short and that for the node farthest from the root to receive the token, it may have to wait 2^ℓ steps.

This approach also circumvents one of the major issues in self-stabilizing spanning tree construction: the spanning tree is built here from the root toward the other nodes, and the root never waits for information from the spanning tree – typically expected from leaves up to the root – thus avoiding deadlock when no leaves exist (*i.e.*, when only cycles map the network).

Although our algorithm is designed so that the distance modulo 3 of each node is eventually stable (*i.e.*, eventually remains unchanged), Algorithm 1 is not silent. Indeed, even when the distributed version of $BFS^{\equiv 3}$ is built, an underlying token-passing mechanism keeps operating indefinitely.

3.2 Variables

Each node v maintains three variables. A variable that, upon stabilization of the algorithm, stores the distance to the root modulo 3, this variable, called the **rank**, is denoted by k. Consequently, for each node v, the variable k_v takes values in $\{0, 1, 2, \perp\}$. As explained in the sequel, the extra value $\perp$ is used to handle errors. When the system is stabilized, *i.e.*, Variable k_v is supposed to induce a kinship relationship matching $BFS^{\equiv 3}$. However, in the arbitrary initial configuration, this variable may be incorrect, affecting the relationship that should normally exist to form $BFS^{\equiv 3}$. Hence, cycles (induced by the parent relationships in the graph G) and incorrect DAGs (*i.e.*, DAGs rooted on nodes other than the root, or with multiple roots) may exist in an initial configuration.

Therefore, we add a token diffusion mechanism to stabilize k_v. The circulation of tokens is handled by the variable t. This variable takes values in $\{true, false, wait\}$. The states *true* and *false* indicate whether a node holds a token or not. The *wait* state serves two purposes: it prevents a node that already holds a token from immediately receiving another token, and it informs a token-holding node about which node(s) sent it the token. After stabilization, nodes cycle through the states *false*, *true*, and *wait*, in that order. A node with a *true* token moves to a *wait* token in the subsequent step, while a node with a *wait* token moves to a *false* token in the subsequent step. A node may remain in the *false* state for more than one step.

The last local variable, called **bit**, and denoted by b, is used to construct binary words originating at the root. For each node v, the variable b_v can take values in $\{\mathbf{0}, \mathbf{1}, \top\}$. The states **0** and **1** are the normal states of a bit, and as explained further, $\top$ is used to manage a reset mechanism when an error is detected locally.

Note that those three variables are also present at the root r. But, they are considered as constant with the following values: $k_r = 0$, $t_r = true$, and $b_r = \mathbf{1}$. The root has no rules to execute. Note that since the root has $t_r = true$ and $b_r = \mathbf{1}$, it *continuously* proposes a token to its neighbors.

3.3 Parent-child relationships sets and predicates

In the following, we denote by $N(v)$ the set of neighbors of node v, and by $pt_{(v,u)}$ the port number of v leading to u. The parents relationship is defined by the rank variable (*i.e.,* k) of each node. Let us denote by $P(v)$ the set of v's parents: $P(v) = \{u \in N(v) | k_u = (k_v - 1) \mod 3\}$.

To obtain a BFS spanning tree, all nodes must choose a single parent among its parents' set $P(v)$. We use the minimum port number to break such symmetric cases.

$$p(v) = \begin{cases} u | w \in P(v) \wedge pt_{(v,u)} = \min\{pt_{(v,w)}\} & \text{if } P(v) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

Children of node v, noted $C(v)$, are formally defined as: $C(v) = \{u \in N(v) | k_u = (k_v + 1) \mod 3\}$

Let $S(v)$ be the set of siblings of node v defined as follow: $S(v) = \{u \in N(v) | k_u = k_v\}$

3.4 Algorithm `TokBin`

Algorithm 1 Algorithm `TokBin` for Node $v \in V \setminus \{r\}$.

R_{er}	:	$b_v \neq \top \wedge Er(v) \longrightarrow t_v := wait; b_v := \top$
R_{reset}	:	$k_v \neq \bot \wedge b_v = \top \longrightarrow k_v := \bot; t_v := false; b_v := \mathbf{0}$
R_{erRank}	:	$\neg Er(v) \wedge k_v \neq \bot \wedge b_v \neq \top \wedge t_v = false \wedge TakeO(v) \longrightarrow t_v := wait; b_v := \top$
R_{join}	:	$\neg Er(v) \wedge k_v = \bot \wedge TakeR(v) \longrightarrow k_v := k_t(v); b_v := \mathbf{1}$
R_{tok}	:	$\neg Er(v) \wedge k_v \neq \bot \wedge t_v = false \wedge TakeP(v) \longrightarrow t_v := true$
R_{add}	:	$\neg Er(v) \wedge t_v = true \longrightarrow t_v := wait; b_v := (b_v + 1) \mod 2$
R_{ready}	:	$\neg Er(v) \wedge t_v = wait \longrightarrow t_v := false$

Our algorithm `TokBin` (see Algorithm 1) is composed of seven rules. Note that our approach is not silent: once the system has stabilized, rules $\underline{R_{tok}}$, $\underline{R_{add}}$, and $\underline{R_{ready}}$ are activated infinitely often by all nodes in the network. The others rules are executed in the stabilizing phase only.

Predicates, Macros, and Functions used in Algorithm 1 are formally defined in Subsections 3.4.1 and 3.4.2.

We use the state $\top$ of variable b_v to indicate that a node v is in an error state. Consequently, rule $\underline{R_{er}}$ detects local errors: a node v that is not in error yet observes an error (predicate $Er(v)$) activates this rule to transition to an error state. In the next step, a node already in an error state resets itself via rule $\underline{R_{reset}}$.

For our algorithm to function correctly, it is necessary to clean as much as possible of the network when an error is detected. So, when an error is detected when executing rule $\underline{R_{er}}$, the predicate $Er(v)$ of its neighbors also becomes true, allowing the the error to propagate when the neighbors also execute rule $\underline{R_{er}}$. This process may not propagate to the entire network if the root is an articulation point of the graph (that is, the graph minus the root node G^* is disconnected). However, the error does propagate to the entire connected component of G^*.

Rules $\underline{R_{tok}}$, $\underline{R_{add}}$, and $\underline{R_{ready}}$ govern token circulation. The system is synchronous, and once the algorithm converges, node ranks remain unchanged. Therefore, when tokens circulate, a node v must receive its token(s) from every parent u satisfying $b_u = 1$. These conditions are captured by the predicate $TakeP(v)$, described formally later. If $TakeP(v)$ is true, v executes rule $\underline{R_{tok}}$ to acquire a token (possibly merging its parents' tokens). When a node has a token, its children must retrieve it if its bit is **1**; otherwise, the token disappears. In either case, the node holding the token releases it by executing rule $\underline{R_{add}}$, which also increments its b variable. Note that, to enable neighbors with tokens to detect which nodes have sent one token, rule $\underline{R_{add}}$ sets the token variable to *wait* rather than directly setting it to $false$. This delay simplifies the proof of the algorithm. Finally, after sending a token and incrementing its variable b_v, node v can execute rule $\underline{R_{ready}}$ to indicate that it is ready to receive a new token. Figure 1 illustrates a possible execution of these three rules.

Observe that a node v must receive token(s) from all of its parents to maintain the parent-child relationships; otherwise, v detects an error. This error is captured by predicate $TakeO(v)$, described formally below. In such a case, v executes rule $\underline{R_{erRank}}$ to declare itself in an error state.

The last rule $\underline{R_{join}}$, concerns nodes that have been reset, and therefore do not hold a rank. A reset node v that is offered a token by a set of nodes P' – all sharing the same rank (predicate $TakeR(v)$) – joins the spanning structure by taking the nodes in P' as its parents. To do so, v adjusts its rank based on the nodes in P' and also acquires a token.

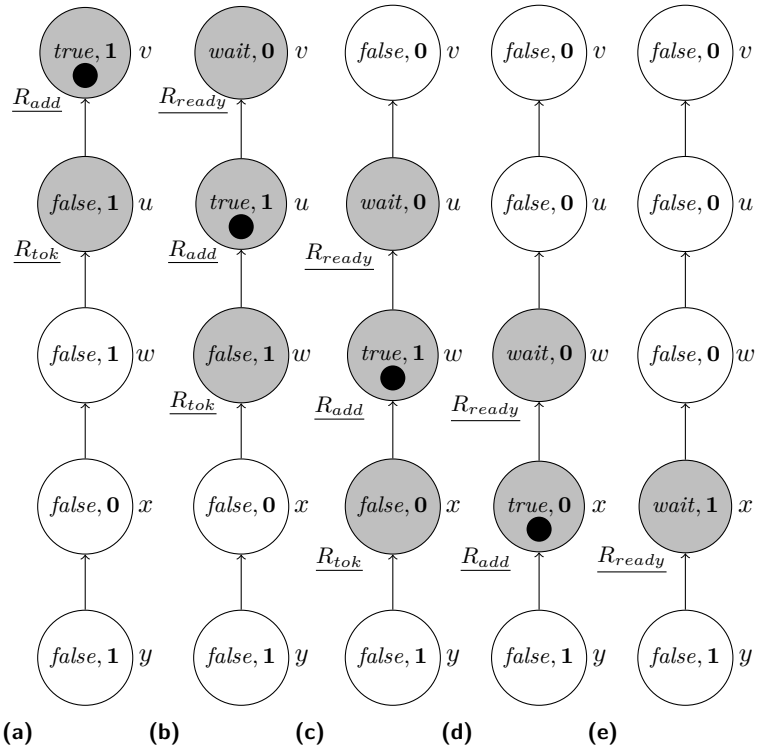


Figure 1 The gray nodes in each subfigure represent activatable nodes. (a) Node v possesses and offers a token. Node u acquires the token by applying the rule $\underline{R_{tok}}$, and node v increments its bit using $\underline{R_{add}}$. (b) Node v applies rule $\underline{R_{ready}}$ to indicate that it is ready to receive a new token. Node u holds the token, it releases the token and increments its bit by applying $\underline{R_{add}}$. Simultaneously, the node w executes rule $\underline{R_{tok}}$ to acquire the token. (d) Node x holds the token; however, node y does not accept it because node x has $b_x = \mathbf{0}$.

3.4.1 Token reception sets and predicates

In the remainder, any node satisfying predicate $Reset(v)$ is referred to as a reset node:

$$Reset(v) \equiv k_v = \perp \wedge t_v = false \wedge b_v = \mathbf{0} \tag{1}$$

In our approach, a node v may receive tokens from a neighbor u if and only if the neighbor's binary variable satisfies $b_u = \mathbf{1}$. The set $Tok(v)$ represents the neighbors of v that may send a token to v.

$$Tok(v) = \{u \in N(v) | t_u = true \wedge b_u = \mathbf{1}\} \tag{2}$$

For our purpose, a distinction is made based on which node transmits the token to v. The two following predicates apply when node v has a valid rank (*i.e.,* $k_v \neq \perp$), allowing it to identify its parent set. The predicate $TakeP(v)$ holds if all proposed tokens are from all parents of v, while predicate $TakeO(v)$ holds if at least one token sender is not the parent of v, or if not all parents of v propose a token.

$$TakeP(v) \equiv Tok(v) \neq \emptyset \wedge Tok(v) = P(v) \tag{3}$$

$$TakeO(v) \equiv Tok(v) \neq \emptyset \wedge Tok(v) \neq P(v) \tag{4}$$

The final predicate in this section is used by a reset node v. It verifies that all token sender nodes share the same rank, and that no node outside the token sender set possesses that same rank.

$$Tok_{rk}(v) = \min\{k_u | u \in Tok(v)\}$$

$$TakeR(v) \equiv Tok(v) \neq \emptyset \wedge (\forall u, w \in Tok(v) | k_u = k_w) \wedge (\forall u \in N(v) \setminus Tok(v) | k_u \neq Tok_{rk}(v)) \tag{5}$$

When a new reset node v satisfies predicate $TakeR(v)$, it can join the spanning structure by becoming a child of nodes in $Tok(v)$. In doing so, v adjust its rank accordingly, using function $k_t(v)$:

$$k_t(v) = (\{k_u | u \in Tok(v)\} + 1) \mod 3 \tag{6}$$

3.4.2 Error Detection sets and predicates

Let us introduce some predicates for all nodes in $V \setminus \{r\}$. Inconsistencies between variables can arise due to a corrupted initial configuration, necessitating detection by the algorithm. A node v with an invalid rank ($k_v = \perp$) is considered a reset node, and thus all its variables must reflect the state of a reset node. When a node v detects an error, it sets $b_v = \top$. After detecting an error, the decision is made to clean the network. Consequently, any non-reset node that has an erroneous neighbor transitions to an error state. In other words, errors propagate from a node to its neighborhood, as captured by the following predicate.

$$Er_{prg}(v) \equiv \exists u \in N(v) | k_u \neq \perp \wedge b_v = \top \tag{7}$$

Predicate $Er_{var}(v)$ detects inconsistencies between variables of a reset node.

$$Er_{var}(v) \equiv \neg Er_{prg}(v) \wedge b_v \neq \top \wedge (k_v = \perp \wedge (t_v \neq false \vee b_v \neq 0)) \tag{8}$$

If a node v has at least one child with a token, and v has not sent a token (i.e., $\neg(t_v = wait \wedge b_v = \mathbf{0})$), then v detects an error thanks to predicate $Er_{tp}(v)$.

$$Er_{tp}(v) \equiv \neg Er_{prg}(v) \wedge b_v \neq \top \wedge C(v) \neq \emptyset \wedge (\exists u \in C(v) | t_u = true) \wedge \neg(t_v = wait \wedge b_v = \mathbf{0}) \quad (9)$$

We deal with an synchronous scheduler, and we construct a $BFS^{\equiv 3}$. As a consequence, all parents of a node v must have the same state (see predicate $ErP(v)$), all children of v must share the same state (see predicate $ErC(v)$), and all siblings of v must be in the same state as v (see predicate $ErS(v)$). Moreover, a node that has a valid rank, children and no token must not have any reset nodes as neighbors (see predicate $Rt_nd(v)$). In a normal execution of the algorithm, when a node offers the token, all nodes that do not have a valid rank join it as children. Consequently, a node cannot end up with both children and neighbors lacking a valid rank. All these constrains are captured by the following predicates.

$$ErP(v) \equiv (\exists u, w \in P(v) | b_u \neq b_w \vee t_u \neq t_w) \vee (k_v \neq \perp \wedge P(v) = \emptyset) \quad (10)$$

$$ErC(v) \equiv \exists u, w \in C(v) | b_u \neq b_w \vee t_u \neq t_w \quad (11)$$

$$ErS(v) \equiv \exists u \in S(v) | b_u \neq b_v \vee t_u \neq t_v \quad (12)$$

$$Rt_nd(v) \equiv k_v \neq \perp \wedge C(v) \neq \emptyset \wedge t_v \neq true \wedge (\exists u \in N(v) \mid k_u = \perp) \quad (13)$$

$$Er_N(v) \equiv \neg Er_{prg}(v) \wedge b_v \neq \top \wedge \neg TakeO(v) \wedge \big(ErP(v) \vee ErC(v) \vee ErS(v) \vee Rt_nd(v)\big) \quad (14)$$

A node v with a valid rank ($k_v \neq \perp$) must have a non empty set of parents, otherwise the node is in error:

$$FalseR(v) \equiv k_v \neq \perp \wedge P(v) = \emptyset \quad (15)$$

Neighbors u and w of a reset node v with the same rank must have identical values for variables b and t. Otherwise, v detects an error and enters the error state (i.e., $b_v = \top$) so that, in the next step, nodes u and w can also transition to the error state. Predicate $Er_\pi(v)$ is dedicated to this purpose:

$$Er_\pi(v) \equiv Reset(v) \wedge (\forall u, w \in N(v) | k_u = k_w \wedge (b_u \neq b_w \vee t_u \neq t_w)) \quad (16)$$

A node v can then apply the rest of the rules of the algorithm if it does not find an error in its neighborhood, and it is not in an error state itself. Predicate $Er(v)$ captures this.

$$Er(v) \equiv Er_{var}(v) \vee Er_{tp}(v) \vee Er_N(v) \vee FalseR(v) \vee Er_{prg}(v) \vee Er_\pi(v) \quad (17)$$

4 Main Result

We now state our main technical result.

▶ **Theorem 1.** *In a semi-uniform model where r is the distinguished node, and every other node is anonymous, our synchronous deterministic self-stabilizing algorithm* `TokBin` *constructs a BFS tree rooted in r using $O(1)$ bits per node in $O(2^\varepsilon)$ steps, where ε denotes the eccentricity of r.*

Overview of Correctness. To establish space complexity, we simply observe that each node $v \in V$ maintains three local variables: k_v, t_v, and b_v. Together, those variables require only 6 bits, ensuring a per-node space complexity of $O(1)$ bits. The correctness proof is carried out in several stages. First, we demonstrate – using a potential function – that starting from an arbitrary configuration, our system converges and remains within a subset of configurations denoted by $\Gamma_{\bar{e}}$. More precisely, $\Gamma_{\bar{e}}$ is the set of configurations where obvious errors $Er_{var}(\gamma_0, v) = true$, $Er_{tp}(\gamma_0, v) = true$, or $Er_N(\gamma_0, v) = true$ (see Equations 8, 9, and 14) are no longer present.

Afterwards, we define the set of legal configurations Γ^*. A configuration is considered legal if, for every node v, v's rank is equal to its distance from the root in the graph modulo 3. Moreover, we must ensure that the nodes' ranks no longer change. In our synchronous setting, this requires that all shortest paths from the root to any node v are identical with respect to the nodes' states on the paths. In other words, every node v at the same distance from the root must be in the same state (*i.e.*, has identical values for variables t_v and b_v).

Then, we need to prove that, *(i)* starting from a possibly illegal (but free of obvious errors) configuration in $\Gamma_{\bar{e}}$, the system converges to a legal configuration, and *(ii)* the system remains in the set of legal configurations Γ^*, thanks to our algorithm. Note that in $\Gamma_{\bar{e}}$, only the rules $\underline{R_{tok}}$, $\underline{R_{add}}$, and $\underline{R_{ready}}$ are executable – these are the rules that govern token circulation.

The second part of the proof (closure) is established by observing that at each step of the algorithm execution, the predicate defining legal configurations is maintained. The first par of the proof (convergence) is more involved, and is broken into several steps.

The remainder of the correctness proof is as follows. First, we show that within $O(n)$ steps, all tokens present in the initial configuration are destroyed, resulting in configurations belonging to the set Γ_{tr}. A key component in achieving this is the predicate $Er_{tp}(v)$ (see Equation 9) and the rule $\underline{R_{add}}$, which modifies the variable b. Recall that a node can receive the token if and only if its parent p has $b_p = \mathbf{1}$. Therefore, if a node v receives the token when its parent offers it (*i.e.*, when $b_p = \mathbf{1}$), then the next time the parent offers it, v cannot accept the token since the parent has $b_p = 0$.

Now we can prove that, starting from a configuration in Γ_{tr}, if a node receives the token, it must have a legal rank (*i.e.*, $k_v = d_v^r \mod 3$). Next, we formalize the paths originating from the root to demonstrate that any node v whose every path from the root is legal receives the token in at most $O(2^{d_v^r - 1})$ steps, where d_v^r is the distance from the root r to v in the graph (d_v^r is bounded by ε).

With these results, we have all the necessary tools to prove that our system converges to a set of error-free configurations, denoted by Γ_{cl}. More precisely, Γ_{cl} comprises configurations where nodes are either legal or have been restarted. Finally, starting from a configuration in Γ_{cl}, we prove convergence toward the set of legal configurations Γ^*.

5 Concluding Remarks

We presented the first constant space deterministic self-stabilizing BFS tree construction algorithm that operate in semi-uniform synchronous networks of arbitrary topology. Our solution `TokBin` is independent of any global network parameter (maximum degree, diameter, number of nodes, etc.). As classical techniques such as point-to-neighbor and distance-to-the-root cannot be used in such a constrained setting, we introduced a novel infinite token stream technique to break cycles and restore BFS construction that may be of independent interest for other tasks. We briefly mention how our BFS construction can be used to optimally color

bipartite graphs (that is, with two colors BLACK and WHITE). All nodes run `TokBin` so that eventually a $BFS^{\equiv 3}$ is constructed. With respect to colors, the root has color BLACK and retains it forever, while every other node take the opposite color of its parents whenever it executes `TokBin` (if it has parents). After stabilization of the $BFS^{\equiv 3}$ construction, all children of the root take the same color, and after one step their children take the root color, and so on, so that a 2-coloring of the bipartite graph is eventually achieved. This very simple application of `TokBin` still uses constant memory, as opposed to dedicated previous solutions to the same problem found in the literature [29, 28] that require $\Omega(\log n)$ bits per node.

Two important open problems remain. First, our solution heavily relies on the network operation being fully synchronous (all nodes execute code in a lock-step synchronous fashion). It would be interesting to extend our token stream technique to an asynchronous setting. In this setting, we expect that some synchronization technique will be necessary, and it remains unclear whether such additional mechanism can be done in constant space. Second, while our algorithm uses a constant amount of memory per node, its time complexity (measured in steps) is exponential. This is due to the exponential decay mechanism for tokens that may require the root to send an exponential number of them to finally reach the furthest nodes in the network. One could speed up this process by adding more states to the b variables: if the domain of b becomes $\{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{k}, \top\}$, and token are accepted for any value that is not $\mathbf{0}$ or $\top$ (assuming b is incremented modulo k anytime the incoming token is accepted), less tokens are destroyed, but the overall time complexity remains exponential. By contrast, solutions that make use of $O(\log \Delta + \log \log n)$ bits per node have polynomial time complexity [10]. Whether there exists a constant space and polynomial time solution to the problem is left for future research.

References

1 Yehuda Afek and Anat Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. *Chic. J. Theor. Comput. Sci.*, 1998, 1998. URL: `http://cjtcs.cs.uchicago.edu/articles/1998/3/contents.html`.

2 Yehuda Afek, Shay Kutten, and Moti Yung. Memory-efficient self stabilizing protocols for general networks. In Jan van Leeuwen and Nicola Santoro, editors, *Distributed Algorithms, 4th International Workshop, WDAG '90, Bari, Italy, September 24-26, 1990, Proceedings*, volume 486 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 1990. `doi:10.1007/3-540-54099-7_2`.

3 Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit, editors. *Introduction to Distributed Self-Stabilizing Algorithms*. Synthesis Lectures on Distributed Computing. Morgan & Claypool Publishers, 2019.

4 Anish Arora and Mohamed G. Gouda. Distributed reset. *IEEE Trans. Computers*, 43(9):1026–1038, 1994. `doi:10.1109/12.312126`.

5 James Aspnes. Distributed breadth first search, 2005. Lecture notes of Distributed Computing, Fall 2005, http://pine.cs.yale.edu/pinewiki/DistributedBreadthFirstSearch.

6 Baruch Awerbuch and Rafail Ostrovsky. Memory-efficient and self-stabilizing network {RESET} (extended abstract). In James H. Anderson, David Peleg, and Elizabeth Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 254–263. ACM, 1994. `doi:10.1145/197917.198104`.

7 Lélia Blin, Shlomi Dolev, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. Fast self-stabilizing minimum spanning tree construction - using compact nearest common ancestor labeling scheme. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September*

13-15, 2010. Proceedings, volume 6343 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2010. doi:10.1007/978-3-642-15763-9_46.

8 Lélia Blin, Laurent Feuilloley, and Gabriel Le Bouder. Optimal space lower bound for deterministic self-stabilizing leader election algorithms. *Discret. Math. Theor. Comput. Sci.*, 25(1), 2023. doi:10.46298/DMTCS.9335.

9 Lélia Blin, Maria Potop-Butucaru, and Stephane Rovedakis. A super-stabilizing log(n)log(n)-approximation algorithm for dynamic steiner trees. *Theor. Comput. Sci.*, 500:90–112, 2013. doi:10.1016/J.TCS.2013.07.003.

10 Lélia Blin and Sébastien Tixeuil. Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. *Distributed Computing*, 31(2):139–166, 2018. doi:10.1007/s00446-017-0294-2.

11 Lélia Blin and Sébastien Tixeuil. Compact self-stabilizing leader election for general networks. *J. Parallel Distributed Comput.*, 144:278–294, 2020. doi:10.1016/J.JPDC.2020.05.019.

12 Christian Boulinier, Ajoy Kumar Datta, Lawrence L. Larmore, and Franck Petit. Time and space optimal distributed BFS tree construction. *Information Processing Letters*, 108(5):273–278, 2008. doi:10.1016/j.ipl.2008.05.016.

13 NS Chen, HP Yu, and ST Huang. A self-stabilizing algorithm for constructing spanning trees. *Information Processing Letters*, 39:147–151, 1991. doi:10.1016/0020-0190(91)90111-T.

14 Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, 2008. doi:10.1145/1383369.1383373.

15 Zeev Collin and Shlomi Dolev. Self-stabilizing depth-first search. *Inf. Process. Lett.*, 49(6):297–301, 1994. doi:10.1016/0020-0190(94)90103-1.

16 Ajoy K. Datta, Stéphane Devismes, Colette Johnen, and Lawrence L. Larmore. Analysis of a memory-efficient self-stabilizing BFS spanning tree construction. *Theor. Comput. Sci.*, 955:113804, 2023. doi:10.1016/J.TCS.2023.113804.

17 Ajoy Kumar Datta, Colette Johnen, Franck Petit, and Vincent Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Comput.*, 13(4):207–218, 2000. doi:10.1007/PL00008919.

18 Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.

19 S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7:3–16, 1993. doi:10.1007/BF02278851.

20 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

21 Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999. doi:10.1007/S002360050180.

22 Mohamed G. Gouda, Jorge Arturo Cobb, and Chin-Tser Huang. Fault masking in tri-redundant systems. In Ajoy Kumar Datta and Maria Gradinariu, editors, *Stabilization, Safety, and Security of Distributed Systems, 8th International Symposium, SSS 2006, Dallas, TX, USA, November 17-19, 2006, Proceedings*, volume 4280 of *Lecture Notes in Computer Science*, pages 304–313. Springer, 2006. doi:10.1007/978-3-540-49823-0_21.

23 Ted Herman. Origin of self-stabilization. In Krzysztof R. Apt and Tony Hoare, editors, *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, volume 45 of *ACM Books*, pages 81–104. ACM / Morgan & Claypool, 2022. doi:10.1145/3544585.3544592.

24 Ted Herman and Sriram V. Pemmaraju. Error-detecting codes and fault-containing self-stabilization. *Inf. Process. Lett.*, 73(1-2):41–46, 2000. doi:10.1016/S0020-0190(99)00164-7.

25 Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 226–239. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365691.

26 C Johnen. Memory-efficient self-stabilizing algorithm to construct BFS spanning trees. In *Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 125–140. Carleton University Press, 1997.

27 Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self stabilizing verification, computation, and fault detection of an MST. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 311–320. ACM, 2011. doi:10.1145/1993806.1993866.

28 Adrian Kosowski and Lukasz Kuszner. Self-stabilizing algorithms for graph coloring with improved performance guarantees. In Leszek Rutkowski, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada, editors, *Artificial Intelligence and Soft Computing - ICAISC 2006, 8th International Conference, Zakopane, Poland, June 25-29, 2006, Proceedings*, volume 4029 of *Lecture Notes in Computer Science*, pages 1150–1159. Springer, 2006. doi:10.1007/11785231_120.

29 Sumit Sur and Pradip K. Srimani. A self-stabilizing algorithm for coloring bipartite graphs. *Inf. Sci.*, 69(3):219–227, 1993. doi:10.1016/0020-0255(93)90121-2.

Complexity Landscape for Local Certification

Nicolas Bousquet
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Laurent Feuilloley
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Sébastien Zeitoun
CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France

Abstract

An impressive recent line of work has charted the complexity landscape of distributed graph algorithms. For many settings, it has been determined which time complexities exist, and which do not (in the sense that no local problem could have an optimal algorithm with that complexity). In this paper, we initiate the study of the landscape for *space complexity* of distributed graph algorithms. More precisely, we focus on the local certification setting, where a prover assigns certificates to nodes to certify a property, and where the space complexity is measured by the size of the certificates.

Already for anonymous paths and cycles, we unveil a surprising landscape:

- There is a gap between complexity $O(1)$ and $\Theta(\log\log n)$ in paths. This is the first gap established in local certification.
- There exists a property that has complexity $\Theta(\log\log n)$ in paths, a regime that was not known to exist for a natural property.
- There is a gap between complexity $O(1)$ and $\Theta(\log n)$ in cycles, hence a gap that is exponentially larger than for paths.

We then generalize our result for paths to the class of trees. Namely, we show that there is a gap between complexity $O(1)$ and $\Theta(\log\log d)$ in trees, where d is the diameter. We finally describe some settings where there are no gaps at all.

To prove our results we develop a new toolkit, based on various results of automata theory and arithmetic, which is of independent interest.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Local certification, proof-labeling schemes, locally checkable proofs, space complexity, distributed graph algorithms, complexity gap

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.18

Related Version *Full Version*: `https://arxiv.org/abs/2505.20915` [14]

Funding This work is supported by the ANR grant ENEDISC (ANR-24-CE48-7768).

Acknowledgements The authors would like to thank Thomas Colcombet for answering our questions about Chrobak normal form, and the reviewers for useful comments.

1 Introduction

1.1 Approach

Time complexity landscapes for distributed graph algorithms

For distributed graph algorithms, the most classic measure of complexity is *time*, measured by the number of rounds before output. One of the most fruitful research programs on this topic has been the one of charting the complexity landscape [42]. That is, instead of considering specific problems and asking for their time complexity, a long series of papers


© Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun;
licensed under Creative Commons License CC-BY 4.0



have answered the following question: given a complexity function, is there a problem with that complexity? In this paper, we aim at starting the analogue research program for the *space complexity* of distributed graph algorithms.

Let us mention some elements about the time complexity landscape, in order to draw analogies with our setting later. For the sake of simplicity, we only discuss deterministic complexity of locally checkable labelings (LCLs) in bounded degree graphs in the LOCAL model. Before 2016, there were a few of classic complexities: $O(1)$ for very simple tasks, $\Theta(\log^* n)$ for tasks that could be reduced to coloring, and $O(n)$ for global problems (throughout the paper, n refers to the number of nodes in the graph). Then, several seminal papers established that some problems have complexity $\Theta(\log n)$ [15, 21], and that there is no problem whose complexity strictly lies between $\Theta(\log^* n)$ and $\Theta(\log n)$ [21]. This gap sharply contrasts with classic computational theory where the time hierarchy theorem prevents the existence of such gaps [38]. It was also proved that in some intervals infinitely many complexities exist [6, 7, 19, 22], but not *any* complexity function had a corresponding problem. A key strategy in this research area is to understand the landscape for specific classes of networks such as paths [3], grids [16], trees [4, 5, 6], minor-closed graphs [20], etc. In this paper, we will follow a similar approach: characterizing which space complexities are possible depending on the structure of the network.

New direction: A landscape approach to space

The *space* used by distributed algorithms is much less studied than the time, especially if we restrict to distributed graph algorithms. Some models where space is a well-studied measure are population protocols [2], massively parallel computing (MPC) [39] and models similar to Stone Age [26], but they are not relevant to our approach, either because they differ too much from distributed graph computing (having one-to-one communication or a centralized component) or because they fix the space complexity to constant. As far as we know, the only major line of research considering space complexity for distributed graph algorithms is self-stabilization, and more specifically the state model, where the nodes update their states by reading the states of their neighbors [1, 25]. In that model, the algorithm has to cope with transient faults, and consequently it is common to design algorithms (at least implicitly) with two components : one that builds a solution and one that checks the solution, and can reset the system if needed. *Local certification* was introduced to study specifically the space needed for the checking phase, by abstracting the computation of the solution into an oracle, and this is our focus today.

Informally, a local certification of a property is an assignment of labels to the nodes of the graph, such that the nodes can collectively check the correctness of a given property by inspecting the labels in their neighborhoods. This notion is known under different names, depending on the specific model considered: *proof-labeling scheme* [40], locally checkable proofs [36], non-deterministic local decision [33], etc. We will formally define our model in Section 2, and refer to [27] for an introduction to the notion. The (space) complexity of a certification is the maximum size of the label given to a node, and it is known to be basically equal to the space complexity of self-stabilizing algorithms.[1] Finally, since local certification is about decision problems, it is now standard to focus on checking graph properties (instead of solutions to graph problems), and to refer to the set of correct graphs as a *language*.

[1] There are some fine prints to this statement, that we will discuss later, in Section 1.4.

In local certification, the situation is similar to the pre-2016 situation for the time complexity as described above. That is, there are a few well-established complexity regimes – $O(1)$, $\Theta(\log n)$, $\Theta(n)$ and $\Theta(n^2)$ – but no solid explanation as of why these are so common. But unlike for distributed time complexity (for LCLs on trees), we can prove that for a wide range of complexities, there exists a problem with that complexity.

▶ **Theorem 1.** *For general graphs with identifiers, for any non-decreasing function $f(n)$ in $\Omega(\log n)$ and $O(n^2)$, there exists a property that can be certified with $O(f(n))$ bits, but not in $o(f(n))$ bits.*

The proof (deferred to Appendix A) follows a standard construction of [36]. The graphs satisfying the property are the ones made of two copies of a graph H, linked by a path. The graph H is chosen so that it can be encoded on $f(n)$ bits, and a certification consists in giving this encoding to all nodes, so that every node knows H. It is known that this encoding can be checked locally when identifiers are given, hence we get the $O(f(n))$ upper bound. A counting argument allows to prove a matching lower bound [36].

At first sight, Theorem 1 gives a trivial answer to our question about the complexity landscape of local certification. But it relies crucially on several assumptions. First, in the upper bound it is necessary to have unique identifiers to avoid being fooled by symmetries at the checking phase. Also, because of the identifiers and because the value of n needs to be certified, $\Omega(\log n)$ bits are needed. Finally, to make the counting argument of the lower bound work, one cannot restrict too much the graphs in which to apply this theorem. This rises the three following questions that we tackle in this paper.

▶ **Question 2.** *What happens for complexities in $o(\log n)$?*

In other words, is the $\Omega(\log n)$ a limitation of the proof technique, or could there be a gap between constant and $\Theta(\log n)$, as the current state of our knowledge suggests?

▶ **Question 3.** *What about anonymous networks?*

This is especially relevant in conjunction with the previous question, since a unique identifier cannot be encoded in a certificate of $o(\log n)$ bits. Note that even beyond that regime, the impact of the identifiers on local decision is a well-studied topic [31, 30, 32, 29].

▶ **Question 4.** *What about structured graphs, like paths, cycles and trees?*

In the spirit of the work made on the distributed time complexity landscape, we would like to understand local certification on restricted graph classes, in which counting arguments do not apply, or only partially.

1.2 Main results

We start with a simple setting: anonymous paths without input labels. In this setting, a path is characterized by its length, and a language is defined by a set of authorized lengths. Naturally, we assume that the nodes *do not* have the knowledge of n (otherwise the problem is trivial). Given $O(\log n)$ bits, we can recognize any language. This is because the prover can certify the exact length of the path, by giving to every node its distance to a chosen endpoint as a certificate. The nodes can then check locally the correctness of this counter, and the last node can check whether the length belongs to the authorized sizes. On the other hand, with a constant number of bits, we can encode properties like "the path has length $k \mod m$", via counters modulo m (where m is a constant). To build a language whose

complexity would be strictly between these regimes, it is tempting to consider properties of the form: "the path has length k modulo $f(n)$" for arbitrary function f, and for which a counter would use $\log f(n)$ bits. Unfortunately, the nodes are unable to check that the $f(n)$ is correct, since they do not have access to n. This obstacle does not seem easy to bypass and, it is reasonable to conjecture that there is a gap between $O(1)$ and $\Theta(\log n)$. We do establish the existence of a gap, but it only goes up to $\Theta(\log \log n)$.

▶ **Theorem 5.** *Let $c > 1$ and $N \in \mathbb{N}$. Let $\mathcal{P}$ be a property on paths that can be certified with certificates of size $s(n) := \left\lfloor \frac{\log \log n}{c} \right\rfloor$ for all $n \geqslant N$. Then, $\mathcal{P}$ can also be certified with constant-size certificates.*

This is *the first gap established in local certification*. At first, this $\log \log n$ looks like an artifact of the proof but, surprisingly, it is not! We next prove that there exists a language for which the optimal certificate size is $\Theta(\log \log n)$. (Given the previous theorem, it is sufficient to prove that this language can be certified with $O(\log \log n)$ bits but not with $O(1)$ bits.)

▶ **Theorem 6.** *There exist properties on paths that can be certified with certificates of size $O(\log \log n)$, but not with certificates of size $O(1)$.*

It is *the first time that this regime appears in the area of local certification* (here under the promise that the graph is a path). The language we use is the set of paths whose length is not a product of consecutive primes; we will come back to it.

Now, moving on to cycles, there is a second surprise. We expect to see the same landscape in paths and cycles, but the intermediate regime actually disappears in cycles, and there is a gap between $O(1)$ and $\Theta(\log n)$.

▶ **Theorem 7.** *Let $c > 12$ and $N \in \mathbb{N}$. Let $\mathcal{P}$ be a property on cycles that can be certified with certificates of size $s(n) := \left\lfloor \frac{\log n}{c} \right\rfloor$ for every integer $n \geqslant N$. Then, $\mathcal{P}$ can also be certified with constant-size certificates.*

Finally, we study the case of trees, where the picture is more complex, as it depends on the exact setting considered. In some settings, we can prove that there is no gap (any reasonable function corresponds to the optimal certificate size for a language), and in some other cases, the gap from paths persists. These settings depend on three parameters: (1) whether we consider certificate size as a function of the number of nodes n or of the diameter d, (2) whether the nodes can only see what is at distance 1 or at larger distance (in other words the verification radius is 1 or larger), and (3) whether the maximum degree is bounded or not. We establish a classification of these different settings. In particular, we prove that *the gap for paths is generalized to arbitrary trees*, when parameterized by the diameter, and the verification radius is 1.

▶ **Theorem 8.** *Let $c > 2$ and $D \in \mathbb{N}$. Let $\mathcal{P}$ be a property in trees (of unbounded degree) that can be certified using $s(d) := \left\lfloor \frac{\log \log d}{c} \right\rfloor$ bits for all $d \geqslant D$ (where d is the diameter). Then, $\mathcal{P}$ can also be certified with constant-size certificates.*

We then show that two assumptions in Theorem 8 are optimal. Namely, we prove in Theorem 9 that we can not hope for a gap in n instead of d in trees (even in caterpillars, that is, paths with leafs attached), and we establish in Theorem 10 that it is essential that the vertices are able to see only at distance only 1, because there is no more gap if the verification radius r is at least 2 (again, even in caterpillars).

▶ **Theorem 9.** *Let $f : \mathbb{N} \to \mathbb{R}$ be a non-decreasing function such that $\lim_{n\to+\infty} f(n) = +\infty$ and for all integers $1 \leqslant s \leqslant t$ we have $f(t) - f(s) \leqslant \log t - \log s$. Then, there exists a property on caterpillars (of unbounded degree) that can be certified with certificates of size $O(f(n))$ and not with certificates of size $o(f(n))$.*

▶ **Theorem 10.** *Let $r > 1$. Let $f : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function such that $\lim_{n\to+\infty} f(n) = +\infty$ and for all integers $1 \leqslant s \leqslant t$ we have $f(t) - f(s) \leqslant \log t - \log s$. Then, if the vertices can see at distance r, there exists a property on caterpillars (of unbounded degree) that can be certified with certificates of size $O(f(d))$ and not with certificates of size $o(f(d))$, where d is the diameter.*

Note that the condition on $f(t) - f(s)$ only ask for a function which is sublogarithmic and whose growth is sublogarithmic, to avoid arbitrarily long plateaus followed by big jumps. This condition is satisfied by all the usual functions ($\sqrt{\log n}$, $\log\log n$, $\log^* n$, etc.).

All the results mentioned so far are in the anonymous setting. We also explore the impact of identifiers and of the knowledge of n on this landscape, but we delay this discussion for now.

1.3 Main techniques

Automata and arithmetic perspectives

To give an overview of our techniques, let us start by describing two perspectives on constant size certification in anonymous paths. As said earlier, a basic building block is to certify with a counter that the length is equal to $i \mod k$ for some constant i and k. One can see this behavior as a run in an automaton with k states inducing a cycle whose initial state being $0 \mod k$ and the final state being $i \mod k$. More generally, any constant size certification can be turned into a finite state automaton. There are various ways to do it, but basically, it consists in having states describing (pairs of) certificates, and transitions that connect states that would be accepted by the local verifier. The existence of an accepting run in the automaton is equivalent to the existence of a certificate assignment making the local verifier accept (see Section 3.1 for a more detailed explanation). (Also see [28], and also [23] for a similar perspective in the context of local problems.) Another point of view is the one of arithmetic. Intuitively, a set of lengths of anonymous paths that are accepted by a constant size certification corresponds to a combination of congruence relations (in other words, such a set is equal to the union of arithmetic progressions up to some point). This point of view allows, for example, to derive that the set of lengths of anonymous paths having a given constant size certification is eventually periodic.

Now, moving on to non-constant certifications, we need to introduce a non-uniform automata model. Indeed, since larger paths imply that we can use larger certificates, it also means larger automata. For each certificate size i, we will have an automaton $\mathcal{A}_i$, and we will require that an incorrect instance is rejected by all these automata, while a correct instance is accepted by at least one (small enough) $\mathcal{A}_i$. If a property has non-constant complexity, then it means that we will need arbitrarily large automata. If there exists a certification of size $s(n)$, then for any instance of length n, the automaton $\mathcal{A}_{s(n)}$ will have to accept. Now from the arithmetic perspective, there is an equivalence between having non-constant complexity and the fact that the set of correct lengths is not eventually periodic.

Establishing gaps in paths and trees

To establish a gap between constant size and $\Theta(\log\log n)$ size in paths, the reasoning is the following.[2] Consider a language S that does not have a constant size certification. For an arbitrary i, we focus on the paths of S that are recognized by $\mathcal{A}_i$ but not by $\mathcal{A}_1, ..., \mathcal{A}_{i-1}$ (assume this is non-empty, which is the case for infinitely many i). This set is itself recognized by an automata: the intersection of $\mathcal{A}_i$ with the complement of the union of $\mathcal{A}_1, ..., \mathcal{A}_{i-1}$. Studying the state complexity of this new automaton (which we do not discuss here) leads us to the fact that it must accept a path of length at most 2^{2^i}. Finally, this upper bound on the minimum length for which some certificate size is needed translates into a lower bound constraint for the optimal complexity, and the double exponential translates into the double logarithm of the theorem.

This proof can be generalized without much modification to labeled paths (that is, paths with inputs) and to larger verification radius (see the full version [14]). This is pretty straightforward, given the previous proof: we use classic automata instead of unary automata, and a slightly different transformation from certification to automata. On the contrary, the proof that gives the right constants for Theorem 5 utilizes Chrobak normal form for unary automata (see full version), and does not generalize easily to the labeled case.

The proof of the analogous gap in trees (Theorem 8), is based on the same insights, but is much more involved. Basically, we adapt our automata to walk in the tree, reading the pending subtrees as labels. Two challenges are that a given tree can be read in many different ways, and that the alphabet is infinite. We argue that the complexity will be captured by the walks that follow a maximum path in the tree, and that the intersection, complement and union that are used in the proof are not harmed by the infinite alphabet.

The $\log\log n$ language

We now turn our attention to Theorem 6, which establishes the existence of a language with optimal certification size $\Theta(\log\log n)$ in anonymous unlabeled paths. Let us start by giving some intuition about how we came to this result. Intuitively, having optimal certificates of size $\Theta(\log\log n)$ means that certificates of size k allow to differentiate between correct and incorrect instances of size order 2^{2^k}. When we implement simple counters modulo some constant using k bits, the modulo is of order 2^k, hence we can distinguish between correct and incorrect instances of size 2^k but not more, in the sense that q and $q+2^k$ will be classified in the same category. Hence we need to do an "exponential jump" in terms of distinguishing capability. We now make two observations. First, given a set of pairs $(r_i, m_i)_{i\leq\ell}$, the fact that the path length is *not equal* to $r_i \mod m_i$ for some $i \leq \ell$ can be certified with certificates of size $O(\max_i \log(m_i))$. Indeed, since it is sufficient to prove that one of the modulo equation is not satisfied, we can simply give explicitly m_i in all the certificates, in addition to the counter modulo m_i. Second, using the Chinese remainder theorem, for any two numbers a and b, there must exist an integer m exponentially smaller than $\max(a, b)$ such that $a \not\equiv b \mod m$. This makes the existence of a $\Theta(\log\log n)$ language believable, but making it into a real construction is another kettle of fish. We end up considering the language of the path whose length *is not* the product of consecutive primes. The scheme consists in certifying that either there is a gap or a repetition in the sequence of primes, via adequate counters, or there is an inconsistency between the largest prime used and what the length of the path is equal to, modulo a well-chosen (small) integer.

[2] Actually this proof gives worse constants than the ones of Theorem 5, but it is the one that generalizes to other settings.

The case of cycles

Theorem 7 states that in cycles there is a gap between constant and logarithmic certification size. This is in sharp contrast with the case of paths, which is surprising given the similarity of the two structures. At a very intuitive level, the difference stems from the fact that the only congruence that we can check in cycles are of the form $0 \mod m$, and not arbitrary $i \mod m$. Indeed, in a path, it is easy to have one endpoint checking counter value zero and the other checking counter value i, while in cycles there are no endpoints. The natural way to simulate the endpoints is to designate a specific vertex v in the cycle to check that the counter starts at 0 from one side and reaches i from the other side, but this is not robust. Indeed there could be more than one designated vertices and still all nodes could accept if each segment is $i \mod m$. This breaks the congruence, except if $i = 0$. This restriction in the congruences prevents us from using the arithmetic tools of the proof of Theorem 6.

Let us now sketch the proof of the gap between certificates of size $O(1)$ and $\Theta(\log n)$ in cycles. We again use an automata-like point of view, but we need to adapt it to work without starting and ending nodes. We consider a sequence of directed certificate graphs[3] $(G_i)_i$. Here, a cycle C with certificates of size i $(c_1, \dots, c_n)$ is accepted by G_i if and only if $(c_1, c_2), (c_2, c_3), \cdots, (c_n, c_1)$ is a closed walk in G_i. For the proof, we consider the first length n_k that is accepted by the k-th graph G_k (corresponding to certificates of size k), but not by smaller ones. Now, if the certificate size is in $o(\log n)$, this walk is much longer than the size of G_k, and therefore it has a lot of cycles. We define a notion of elementary cycle of the graph, and decompose this walk as a linear combination of elementary cycles. If d is the greatest common divisor of these elementary cycles, then $n_k = d \times q$, for some q. We consider a set of numbers of the form $p \times d$, such that (1) p is prime, (2) $p \times d < n_k$ and (3) $p \times d$ is still much larger than the size of G_k. By a generalization of Bézout's identity, (3) implies that all these numbers are accepted by G_k hence they belong to the language. But since they are smaller than n_k, by minimality, they must be recognized by smaller graphs too. We argue by pigeon-hole principle that there must exist one graph $G_{k'}$, $k' < k$ that has two walks of lengths $p_1 d$ and $p_2 d$ that intersect. Then by concatenating portions of the two walks, we can prove that $G_{k'}$ actually accepts all the cycles of size $ap_1 d + bp_2 d$. Finally, using again a generalization of Bézout's identity we can prove that the cycle of length n_k is also recognized by $G_{k'}$, which contradicts the minimality of k.

"No gap" results

We also establish that for several settings there is no gap in the complexity, that is, for any well-behaved function f, there exists a property that has certificates of size $O(f(n))$ but not $o(f(n))$.

Let us sketch the technique of the upper bound for the setting of Theorem 10: restricting to caterpillars, using certificates whose size is a function of d, with verification radius 2. Given a function f, we define a sequence of integers (b_k), such that the k-th term is roughly $f^{-1}(\log k)$. The correct instances are of the following form: a path of length b_k, for some k, such that the i-th node has i leaves, and except for the first one, which has b_k leaves instead of 1. A compact way to certify these instances is to give k to every node. The nodes in the middle check the growth of the number of attached leaves (thanks to their radius 2 verification) and the endpoints check that they have b_k leaves. The diameter is b_k, and the number of bits used is $\log k$ which is basically $f(d)$.

[3] We consider "graphs" and not "automata" here since we have no initial or accepting states, a sequence of certificate only gives us a walk in the graph of pairs of certificates.

1.4 Discussions and open problems

Before we move on to the technical parts, let us discuss open problems and future directions.

Full understanding of paths. For paths, we do not know what happens between $\Theta(\log\log n)$ and $\Theta(\log n)$. By sparsifying the set of primes considered in the $\log\log n$ language (Theorem 6), we can get languages for which the natural upper bound can be positioned in between these two regimes, but Theorem 5 does not provide a matching lower bound anymore, hence we cannot prove that there is no gap.

▶ **Open problem 11.** *Is there a gap between* $\Theta(\log\log n)$ *and* $\Theta(\log n)$ *in paths?*

To solve this question, it would good to get a better understanding of the $\Theta(\log\log n)$ regime, or even a characterization. (For now, we just have one example of a property in this regime.)

General graphs. For general graphs, we prove in the full version that if the radius is larger than 1, then there is no gap. For radius 1, it is unclear whether we should expect the same gap as for trees or not. Our automata-related tools seem too weak to tackle this case (the generalization from trees to graphs in automata theory is notoriously intricate).

▶ **Open problem 12.** *Is there a gap between* $O(1)$ *and* $\Theta(\log\log d)$ *for general graphs? For bounded degree graphs?*

The role of identifiers and of the knowledge of n. Our main results are for the anonymous setting, where the nodes do not have the knowledge of n. In the full version, we explore several settings with identifiers or (approximate) knowledge of n. We can for example prove that a very sharp estimate of n allows to break the $\log\log n$ barrier of Theorem 5, while arbitrarily large identifiers do not help. It is still very unclear how these different assumptions affect the complexity landscape.

Extensions to self-stabilizing algorithms: back space complexity in algorithms. We described in the introduction our wish to chart the space landscape of distributed graph algorithms, and as a first step we focused on local certification. A natural next step is to transfer our results to self-stabilizing algorithms. As mentioned earlier, the two are tightly connected, since the space used for silent self-stabilization is basically captured by the space needed to certify the solution correctness [10]. Actually, [10] implements a transformation from a local certification to a self-stabilizing algorithm, that does require an additive $O(\log n)$ for the memory of the algorithm in comparison to the local certification size. This is usually harmless, but in the setting of this paper, which focuses on sublogarithmic regime, the result is unusable. The restricted topology might allow to shave the additional logarithmic term.

▶ **Open problem 13.** *Are the optimal certificate size and the optimal memory of a self-stabilizing algorithm asymptotically equal in restricted topologies (paths, cycles, trees), even in the sublogarithmic regime?*

When it comes to transferring the sub-log-logarithmic gap for trees to (silent) self-stabilizing algorithms, it would actually be enough to understand whether labelings accepted by tree automata (which are equivalent to monadic second-order on trees) can be built in constant space in a self-stabilizing manner. Incidentally, understanding when one can make this transfer while keeping polynomial time is also a very intriguing question (see [9] for a discussion).

Different landscapes. Theorem 1 establishes that for any super-logarithmic complexity f there exists a property that requires exactly that complexity. But the construction is very unnatural, since the nodes need to know the function f, and the instances are extremely specific. Hence the following question.

▶ **Open problem 14.** *Are there super-logarithmic gaps in the complexity of natural properties, for a reasonable definition of "natural"? What about monadic second-order (MSO) properties?*

It is known that there are properties for which the optimal certification size is $\Theta(n)$, for example having diameter 3 [18, 12] and also $\Theta(n^2)$ [36]. As far as we know the only works about what happens in between are [11] and [41], that have established that for forbidden subgraphs, there are many polynomial complexities, when using verification radius 2.

We mention MSO properties in the open problem because they have received a considerable amount of attention in recent years in local certification [8, 24, 28, 35, 34], and capture many classic properties and problems through logic.

Finally, another research direction is to chart landscapes for other parameters. For example, [13] explored the certification complexity as a function of the maximum degree.

1.5 Organization of the conference version

For this conference version, we focus on the results on paths and cycles, which allows to give most of the insights within the page limit. Some of the proofs of the lemmas and claims are deferred to the appendix. The proof of our more involved results as well as discussions of the knowledge on n and the identifier assumptions only appear in the full version [14].

2 Model and definitions

2.1 Graphs

All the graphs we consider are finite, simple, loopless, connected, and undirected. For completeness, let us recall several basic graph definitions. Let G be a graph. We denote its set of vertices (resp. edges) by $V(G)$ (resp. by $E(G)$), or simply by V (resp. by E) if G is clear from the context. Let $u, v \in V(G)$. The *distance* between u and v, denoted by $d(u, v)$, is the smallest number of edges in a path from u to v. The *diameter* of G is the largest distance between any two vertices. If G is a path, its *length* is its number of vertices (or equivalently, its diameter plus one). We say that G is a *caterpillar* if, when removing all the degree-1 vertices in G, the resulting graph is a path, called the *central path* of G (which is induced by the set of all vertices of degree at least two in G).

2.2 Local certification

Let $G = (V, E)$ be a graph. We will sometimes assume that the vertices of G are equipped with *unique identifiers* and/or with *inputs*. An *identifier assignment* for G is an *injective* mapping from V to some set I (the set of identifiers) and an *input function* is a mapping from V to some set L (the set of labels). If G is equipped with identifiers, we say that we are in the *locally checkable proof model*, else we say that we are in the *anonymous model*. If the vertices of G have inputs, we say that G is *labeled*. Finally, let C be a non-empty set. A *certificate assignment* of G with certificates in C is a mapping $c : V \to C$.

Let $r \geqslant 1$ and c be a certificate assignment for G. Let $u \in V$. The *view of u at distance r* consists in:

- all the vertices at distance at most r from u, and all the edges having at least one endpoint at distance at most $r-1$,
- the restriction of c to these vertices,
- the restriction of the identifier assignment (if any) and of the input function (if any) to these vertices.

A *verification algorithm (at distance r)* is a function taking as input the view at distance r of a vertex, and outputting a decision, *accept* or *reject*. In all this paper, if r is not mentioned in a statement of a result, it is by default equal to 1. When we will consider settings where $r \geqslant 2$, it will always be explicitly written.

Let $\mathcal{C}$ be a class of (possibly labeled) graphs and $\mathcal{P}$ be a property on graphs in $\mathcal{C}$. Note that if we consider labeled graphs, the fact that a graph $G \in \mathcal{C}$ satisfies the property $\mathcal{P}$ does not depend only on the structure of G: it depends also on its input function. In other words, it is possible that a graph $G \in \mathcal{C}$ satisfies $\mathcal{P}$ and that another graph $G' \in \mathcal{C}$ with the same structure as G but with a different input function does not satisfy $\mathcal{P}$. Let $s : \mathbb{N} \to \mathbb{N}$. We say that there exists a *certification scheme for $\mathcal{P}$* with certificates of size s if there exists a verification algorithm such that the two following conditions are satisfied:

- (Completeness) For every n-vertex graph $G \in \mathcal{C}$ that satisfies $\mathcal{P}$, and for every identifier assignment of G (if we are in the locally checkable proof model), there exists a certificate assignment in $\{0, \dots, 2^{s(n)} - 1\}$ such that the verification of every vertex accepts (we say that the graph is globally accepted).
- (Soundness) For every graph $G \in \mathcal{C}$ that does not satisfy $\mathcal{P}$, for every identifier assignment of G (if we are in the locally checkable proof model), for every $k \in \mathbb{N}$ and every certificate assignment in $\{0, \dots, 2^k - 1\}$, at least one vertex rejects.

Let us emphasize that, if G does not satisfy $\mathcal{P}$, then for any assignment of certificates *of any size*, at least one vertex rejects. Let us also point out the fact that in a certification scheme for a property $\mathcal{P}$ in some class $\mathcal{C}$ (in this paper, we will for instance consider the cases where $\mathcal{C}$ is the class of paths, of cycles, of trees...), the vertices have the promise that the graph belongs to $\mathcal{C}$. In other words, the certification scheme depends on the property $\mathcal{P}$ *and* on the class $\mathcal{C}$, and we are not concerned by the output of the verification procedure of the vertices in graphs that do not belong to $\mathcal{C}$.

3 Gap between $O(1)$ and $\Theta(\log \log n)$ in paths

The goal of this section is to prove the following result:

▶ **Theorem 15.** *Let $c > 2$ and $N \in \mathbb{N}$. Let $\mathcal{P}$ be a property on paths that can be certified with certificates of size $s(n) := \left\lfloor \frac{\log \log n}{c} \right\rfloor$ for all $n \geqslant N$. Then, $\mathcal{P}$ can also be certified with constant-size certificates.*

Note that the constant c is larger here than in Theorem 5. We prove the stronger version in the full version.

3.1 Preliminary: automata point of view

For every property $\mathcal{P}$ on paths, we can associate a subsets S of integers such that a path is accepted if and only if its length is in S. The property $\mathcal{P}$ is equivalent to S and in the rest of the proof, we will completely forget the property $\mathcal{P}$ and only focus on S. For every $k \in \mathbb{N}$, let us denote by C_k the set of certificates of size k, and by S_k the set of lengths of the paths that are accepted with certificates in C_k. We have: $S = \bigcup_{k \in \mathbb{N}} S_k$.

The set S_k is a regular language that is accepted with the following nondeterministic finite automaton $\mathcal{A}_k$ over a unary alphabet. The set of states is the set of pairs of certificates of size k plus two additional states, i and f, that is: $C_k^2 \cup \{i, f\}$. There is a single initial state which is i and a single final state which is f. The transitions are the following:

- for every $c_1, c_2 \in C_k$, we put a transition between states i and (c_1, c_2) if a vertex of degree 1 that has certificate c_1 and has a neighbor with certificate c_2 accepts;
- for every $c_1, c_2, c_3 \in C_k$, we put a transition between states (c_1, c_2) and (c_2, c_3) if a vertex of degree 2 that has certificate c_2 and has two neighbors with certificates c_1 and c_3 accepts;
- for every $c_1, c_2 \in C_k$, we put a transition between states (c_1, c_2) and f if a vertex of degree 1 that has certificate c_2 and has a neighbor with certificate c_1 accepts;
- if there exists $c \in C_k$ such that an isolated vertex with certificate c accepts, we put a transition from i to f.

Let us give an example to make things more concrete. Assume that we want to certify that the length of a path is divisible by 3. There is an easy way to do it by using three certificates 0, 1, and 2. The prover fixes an endpoint u and for every vertex v, the certificate it gives to v is $d(u, v) \mod 3$. Then, every vertex v checks that one of the two following conditions is satisfied:

- v has degree 1, has certificate 0 or 2, and its neighbor has certificate 1, or
- v has degree 2, and the set of certificates of v and its two neighbors is $\{0, 1, 2\}$.

The automaton corresponding to these certificates is represented on Figure 1.

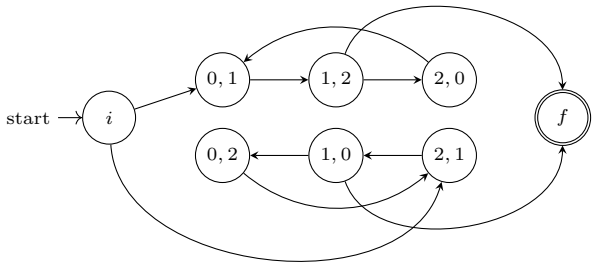


Figure 1 The automaton corresponding to the certificates used to certify that the length of a path is divisible by 3. The states corresponding to the tuples $(0, 0)$, $(1, 1)$ and $(2, 2)$ are not represented because they have no incoming nor outgoing transitions. The final state is the state f.

▶ **Lemma 16.** *For every $t \in \mathbb{N}$, a path on t vertices is accepted with certificates of size k if and only if there exists an accepting run of length t (i.e., going through t transitions) in $\mathcal{A}_k$.*

The intuition behind this lemma has been given earlier and the formal proof is in the full version.

3.2 Proof via state complexity

▶ **Lemma 17.** *Let Σ be a (possibly infinite) alphabet, and let $L_{\mathcal{A}}, L_{\mathcal{B}} \subseteq \Sigma^*$ be two languages over Σ recognized by nondeterministic finite automata $\mathcal{A}$ and $\mathcal{B}$, having $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ states respectively. Then:*

- $L_{\mathcal{A}} \cup L_{\mathcal{B}}$ *can be recognized by an automaton having* $n_{\mathcal{A}} + n_{\mathcal{B}}$ *states*
- $L_{\mathcal{A}} \cap L_{\mathcal{B}}$ *can be recognized by an automaton having* $n_{\mathcal{A}} n_{\mathcal{B}}$ *states*
- $\overline{L_{\mathcal{A}}}$ *can be recognized by an automaton having* $2^{n_{\mathcal{A}}}$ *states*

These statements are folklore, see the full version for explanations.

Proof of Theorem 15. Using the notations introduced previously, the automaton $\mathcal{A}_k$ has $M_k := 2^{2k} + 2$ states and recognizes S_k. Let $\mathcal{P}$ be a property that can not be recognized with constant-size certificates. Then, the set $X \subseteq \mathbb{N}$ containing all the integers $k \in \mathbb{N}$ such that $S_k \not\subseteq \bigcup_{i \leqslant k-1} S_i$ is infinite (this set X contains all the integers k such that there exists a path that can be accepted with certificates of size k but not with smaller size). For $k \in X$, let n_k be the smallest integer in $S_k \setminus \bigcup_{i \leqslant k-1} S_i$. Let $k \in X$ be such that $n_k \geqslant N$ (such an integer $k \in X$ exists because X is infinite and for all distinct $k, k' \in X$ we have $n_k \neq n_{k'}$).

Since $n_k \in S_{s(n_k)}$ and $n_k \notin S_i$ for $i < k$, we have $s(n_k) \geqslant k$. By Lemma 17, $\bigcup_{i \leqslant k-1} S_i$ can be recognized by an automaton that has $\sum_{i=1}^{k-1} M_i \leqslant 2^{2k}$ states. Thus, by Lemma 17, $\overline{\bigcup_{i \leqslant k-1} S_i}$ can be recognized by an automaton that has at most $2^{2^{2k}}$ states. Again by Lemma 17, $S_k \setminus \bigcup_{i \leqslant k-1} S_i$ can be recognized by an automaton having at most $M_k \cdot 2^{2^{2k}}$ states. Since n_k is the smallest integer in $S_k \setminus \bigcup_{i \leqslant k-1} S_i$, it is at most equal to the number of states of this automaton, so it follows that $n_k \leqslant M_k \cdot 2^{2^{2k}} \leqslant 2^{2^{2k+1}}$. Finally we get $s(n_k) \geqslant k \geqslant \frac{\log \log n_k - 1}{2}$, and the result follows. ◀

4 A property with optimal size $\Theta(\log \log n)$ in unlabeled paths

The goal of this Section is to prove the following theorem.

▶ **Theorem 6.** *There exist properties on paths that can be certified with certificates of size* $O(\log \log n)$*, but not with certificates of size* $O(1)$*.*

Before proving Theorem 6, let us show the following result:

▶ **Lemma 18.** *Let* $m, t \in \mathbb{N}$ *and* $m \geqslant 2$*. Certifying that the length a path on* n *vertices satisfies* $n \equiv t \mod m$ *can be done with certificates of size* $O(\log m)$*.*

The proof of this statement can be found in the full version of the paper.

▶ Remark 19. For every $m, t \in \mathbb{N}$ with $m \geqslant 2$, we can also certify that the length n of a path satisfies $n \not\equiv t \mod m$ with certificates of size $O(\log m)$, with the same proof (just by replacing $\mathsf{Distance}[u] = t$ by $\mathsf{Distance}[u] \neq t$ at the end). In particular, with certificates of size $O(\log m)$, we can certify that m divides n, or that m does not divide n.

Finally, let us introduce some notations and give some useful properties. For every $k \geqslant 1$, let us denote by p_k the k-th prime number (i.e. $p_1 = 2, p_2 = 3, p_3 = 5 \ldots$), and let a_k be the product of the k first prime numbers: $a_k := \prod_{i=1}^{k} p_i$. Let $S \subseteq \mathbb{N}$ be the set $\{a_k \mid k \geqslant 1\}$.

▶ **Lemma 20.** *[37] We have* $p_k = \Theta(k \log k)$ *and* $a_k = 2^{k \log k (1 + o(1))}$*.*

▶ **Lemma 21.** *There exists* $c > 0$ *such that, for every even integer* $n \geqslant 2$*, there exists* $k \leqslant c \log n$ *such that* p_k *divides* n *and* p_{k+1} *does not divide* n*.*

Proof. Let n be an even integer and let k be the smallest integer such that p_k divides n and p_{k+1} does not divide n (which exists because n is divisible by $p_1 = 2$). Then, n is divisible by a_k, so $a_k \leqslant n$. Using Claim 20, we get $2^{k \log k (1 + o(1))} \leqslant n$, so $k \log k (1 + o(1)) \leqslant \log n$, and the result follows. ◀

▶ **Lemma 22.** *Let $1 \leqslant s < t$. There exists $m \leqslant \lceil \log t \rceil$ such that $s \not\equiv t \mod m$.*

Proof. By contradiction, assume that for all $m \leqslant \lceil \log t \rceil$, we have $s \equiv t \mod m$. Then, by the Chinese remainder theorem, we get $s \equiv t \mod p$, where p is the least common multiple of $1, 2, \ldots, \lceil \log t \rceil$. It is well-known that this least common multiple is at least t, so we have $1 \leqslant s < t \leqslant p$. Together with $s \equiv t \mod p$, this implies $s = t$, which contradicts the assumption $s < t$. ◀

▶ **Proposition 23.** *Let $n \geqslant 1$, and c be the constant of Lemma 21. Then, $n \notin S$ if and only if at least one of the three following conditions is satisfied:*

1. *n is odd, or*
2. *there exists $1 \leqslant \ell < k \leqslant c \log n$ such that p_ℓ does not divide n and p_k divides n, or*
3. *there exists $1 \leqslant k \leqslant c \log n$ and $1 \leqslant m \leqslant \lceil \log n \rceil$ such that p_k divides n, p_{k+1} does not divide n and $n \not\equiv a_k \mod m$.*

Proof. First, assume that one of the three conditions is satisfied. If condition 1. holds, n is odd, so $n \notin S$ because S contains only even integers. If condition 2. holds, then $n \notin S$ because all the integers in S which are divisible by p_k are also divisible by p_ℓ for all $\ell \leqslant k$. If condition 3. holds, then $n \notin S$, because the only integer in S which is divisible by p_k and not by p_{k+1} is a_k.

Conversely, assume that $n \notin S$, and let us show that at least one of the three conditions is satisfied. Assume that conditions 1. and 2. are not satisfied, and let us show that condition 3. holds. Since condition 1. is not satisfied, n is even, so by Lemma 21, there exists $k \leqslant c \log n$ such that p_k divides n and p_{k+1} does not divide n. Since condition 2. is not satisfied, n is divisible by a_k, so $a_k < n$ (this inequality is strict because, by assumption, $n \notin S$). Finally, by Lemma 22, there exists $m \leqslant \lceil \log n \rceil$ such that $n \not\equiv a_k \mod m$, so condition 3. is satisfied. ◀

Note that the third item is used to avoid accepting numbers with some prime used twice. It is tempting to check this condition directly instead of using our indirect check. But in general this would use integers that are too large, for example in the case where $n = q^2$ with q a prime number.

We are now able to prove Theorem 6.

Proof of Theorem 6. Recall that we assume that the input graph is a path P. Let $\mathcal{P}$ be the property of being a path whose length is *not* in S. First, observe that $\mathcal{P}$ cannot be certified with constant-size certificates. Indeed, properties on paths that can be certified with constant size-certificates are paths whose length is in a set that is eventually periodic (the most simple proof for it uses Chrobak normal form, see the full version), and the set $\overline{S}$ is not. Now, let us show that $\mathcal{P}$ can be certified with certificates of size $O(\log \log n)$.

Let $n \geqslant 1$. If $n \notin S$, the prover certifies that at least one of the three conditions of Proposition 23 is satisfied. More precisely:

- if n is odd, the prover certifies it. By Lemma 18, this needs $O(1)$ bits.
- if there exists $1 \leqslant \ell < k \leqslant c \log n$ such that p_k divides n and p_ℓ does not divide n, the prover writes k and ℓ in the certificate of each vertex, and certifies that p_k divides n and that p_ℓ does not divide n. Since $k \leqslant c \log n$ and $p_k = \Theta(k \log k)$, by Lemma 18 and Remark 19, this needs $O(\log \log n)$ bits.
- if there exists $1 \leqslant k \leqslant c \log n$ and $1 \leqslant m \leqslant \lceil \log n \rceil$ such that p_k divides n, p_{k+1} does not divide n and $n \not\equiv a_k \mod m$, the prover writes k and m in the certificate, certifies that p_k divides n, p_{k+1} does not divides n and that $n \not\equiv a_k \mod m$. By Lemma 18 and Remark 19, this needs $O(\log \log n)$ bits.

The verification of the vertices just consists in checking that the condition given by the prover is indeed satisfied, with the verification procedure of Lemma 18. Note that the vertices do not need to check the bounds on k and m for conditions 2. and 3., because if condition 2. or 3. is satisfied with larger k or m, it also implies that $n \notin S$ (and then $P \in \mathcal{P}$ and P can be also accepted with certificates of size $O(\log \log n)$). These bounds on k and m are only useful to get an upper bound on the size of the certificates. ◀

5 Gap between $O(1)$ and $\Theta(\log n)$ in cycles

In this section we prove the following theorem.

▶ **Theorem 7.** *Let $c > 12$ and $N \in \mathbb{N}$. Let $\mathcal{P}$ be a property on cycles that can be certified with certificates of size $s(n) := \left\lfloor \frac{\log n}{c} \right\rfloor$ for every integer $n \geqslant N$. Then, $\mathcal{P}$ can also be certified with constant-size certificates.*

5.1 Preliminaries on number theory and walks in graphs

Let us give some results from number theory on which we will rely. First, let us recall the prime number theorem:

▶ **Theorem 24** (Prime Number Theorem)**.** *For $n \in \mathbb{N}$, let $\pi(n)$ be the number of prime numbers in $\{1, \ldots, n\}$. Then:*

$$\pi(n) \sim \frac{n}{\ln(n)}$$

From Theorem 24, we can deduce the immediate following corollary:

▶ **Corollary 25.** *Let $c > 12$. For every $n \in \mathbb{N}$, let $\pi_c(n)$ be the number of prime numbers p such that $2^{4n+1} < p \leqslant 2^{n(c/2-2)}$. Then, there exists $n_0 \in \mathbb{N}$ such that for all $n \geqslant n_0$, $\pi_c(n) > n2^{2n}$.*

Proof. For every $n \in \mathbb{N}$, we have $\pi_c(n) = \pi(2^{n(c/2-2)}) - \pi(2^{4n+1})$. Since $c > 12$, by Theorem 24, we have $\pi(2^{4n+1}) = o(\pi(2^{n(c/2-2)}))$. Thus, $\pi_c(n) \sim \pi(2^{n(c/2-2)})$. By applying again the prime number theorem, we get $\pi_c(n) \sim 2^{n(c/2-2)}/(n(c/2-2))$. Thus, $n2^{2n} = o(\pi_c(n))$, and the result follows. ◀

Finally, let us give the following generalization of Bézout's identity that will be useful in the proof of Theorem 7:

▶ **Lemma 26** ([17])**.** *Let $\ell_1, \ldots, \ell_t$ be positive integers. Let $m := \max(\ell_1, \ldots, \ell_t)$ and $d := \gcd(\ell_1, \ldots, \ell_t)$. Then, for every integer $n \geqslant m^2$ which is divisible by d, there exists non-negative integers $a_1, \ldots a_t$ such that $\sum_{i=1}^{t} a_i \ell_i = n$.*

We now move on to some definitions about walks in graphs. A *closed walk* in $\mathcal{G}_k$ is a directed path that begins and ends in the same vertex (it is allowed to pass through the same vertex or the same edge multiple times). The *length* of a closed walk is its number of edges. A closed walk that does not pass through the same vertex twice (except for the starting and ending vertices which are the same) is called an *elementary cycle*. If $\mathcal{C}$ and $\mathcal{C}'$ are two directed closed walks, we say that $\mathcal{C}'$ is a closed subwalk of $\mathcal{C}$ if a subsequence of vertices in $\mathcal{C}$ is equal to $\mathcal{C}'$. See Figure 2 for an example. Note that the length of an elementary closed walk in $\mathcal{G}_k$ is at most equal to the number of vertices of $\mathcal{G}_k$ which is 2^{2k}.

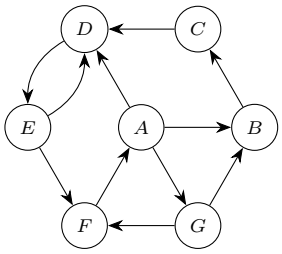


Figure 2 In this directed graph, $ADEDEFA$ is a closed walk of length 6, which is not an elementary cycle. The closed walks $FAGF$ and $CDEFABC$ are elementary cycles and have length 3 and 6 respectively. Moreover, $FAGF$ is a subwalk of $DEFAGFAD$.

5.2 Proof of Theorem 7

All this subsection is devoted to the proof of Theorem 7. The proofs of the claims can be found in Appendix B.

Let $c > 12$ and $N \in \mathbb{N}$. Let S be the set of lengths of cycles in $\mathcal{P}$. Assume by contradiction that there exists a property $\mathcal{P}$ such that, for every integer $n \geqslant N$ satisfying $\mathcal{P}$, the cycle of length n is accepted with certificates of size $s(n)$, where $s(n) \leqslant \left\lfloor \frac{\log n}{c} \right\rfloor$, and that a constant number of certificates are not sufficient to certify $\mathcal{P}$.

For every $k \geqslant 1$, let C_k be the set of certificates of size k, and let S_k be the subset of S corresponding to the cycles accepted with certificates in C_k. Note that $|C_k| = 2^k$, and that we have $S = \bigcup_{k\in\mathbb{N}} S_k$. Let $\mathcal{G}_k = (V_k, E_k)$ be the directed graph where $V_k := C_k^2$ and for all $a, b, c \in C_k$, there is an edge in E_k between (a, b) and (b, c) if and only if a degree-2 vertex with certificate b has a neighbor with certificate a and another neighbor with certificate c accepts (and there are no other edges in E_k, that is there is no edge between (a, b) and (c, d) if $b \neq c$). The directed graph $\mathcal{G}_k$ has 2^{2k} vertices.

▷ Claim 27. For every integer $n \geqslant 3$, $n \in S_k$ if and only there exists a closed walk of length n in $\mathcal{G}_k$.

By Corollary 25, there exists $k_0 \in \mathbb{N}$ such that for all $k \geqslant k_0$, $\pi_c(k) > k2^{2k}$.

Since S is not accepted with a constant number of certificates, the set $X \subseteq \mathbb{N}$ of integers $k \in \mathbb{N}$ such that $S_k \not\subseteq \bigcup_{1\leqslant i<k} S_i$ is infinite. For $k \in X$, let n_k be the smallest integer $S_k \setminus \bigcup_{1\leqslant i<k} S_i$. Finally, let us fix an integer integer $k \in X$, such that $k \geqslant k_0$ and $n_k \geqslant N$ (such an integer $k \in X$ exists because X is infinite and for all distinct $k, k' \in X$ we have $n_k \neq n_{k'}$).

▷ Claim 28. We have $n_k \geqslant 2^{ck}$.

Since $n_k \in S_k$, by Claim 27, there is a closed walk of length n_k in $\mathcal{G}_k$. Let us consider the strongly connected component $\mathcal{G}'_k$ of $\mathcal{G}_k$ containing this closed walk. Let t be the number of elementary cycles in $\mathcal{G}'_k$ that we denote by $\mathcal{C}_1, \ldots, \mathcal{C}_t$, and let $\ell_1, \ldots, \ell_t$ be their lengths. Let $d = \gcd(\ell_1, \ldots, \ell_t)$. We have $d \leqslant 2^{2k}$, because we have $\ell_i \leqslant 2^{2k}$ for every $i \in \{1, \ldots, t\}$ (since $\mathcal{G}_k$ has size 2^{2k}).

▷ Claim 29. Let $\mathcal{C}$ be a closed walk in $\mathcal{G}'_k$, and let ℓ be its length. There exists $b_1, \ldots, b_t \in \mathbb{N}$ such that $\ell = \sum_{i=1}^{t} b_i\ell_i$. Thus, d divides ℓ. In particular, d divides n_k.

▷ Claim 30. Let $m \in \mathbb{N}$ be such that d divides m, and $m \geqslant 2^{4k+1}$. Then, there exists a closed walk in $\mathcal{G}'_k$ of length m. Thus, $m \in S_k$.

Let us now combine these arguments to prove Theorem 7. Before giving the technical details, let us explain the intuition. Let us denote by d the $d := gcd$ of all the lengths of cycles in S_k. Since $\mathcal{G}_k$ has size 2^{2k}, Lemma 30 ensures that all the cycles of length $r \cdot d$ are in $\mathcal{P}$ when r is large enough (but small compared to n_k). Thus there exist many prime numbers p such that pd are in $\mathcal{P}$ and $pd \leq n_k$. By definition of n_k, at least two of them can be certified with the same set of bits and we can obtain a contradiction. Let us now formalize the argument.

Recall that k is an integer such that $k \geqslant k_0$ and $n_k \geqslant N$. Let p be a prime number such that $2^{4k+1} < p \leqslant 2^{k(c/2-2)}$. By Claim 30, we have $pd \in S_k$. Moreover, since $p \leqslant 2^{k(c/2-2)}$ and $d \leqslant 2^{2k}$, we have $pd \leqslant 2^{kc/2}$, that is, $pd \leq \sqrt{n_k}$ using Claim 28. Since n_k is the smallest integer in $S_k \setminus \bigcup_{1 \leqslant i < k} S_i$, we have $pd \in \bigcup_{1 \leqslant i < k} S_i$. For every $i \in \{1, \dots, k-1\}$, let X_i be the set of prime numbers $p \in \{2^{4k+1}+1, \dots, 2^{k(c/2-2)}\}$ such that $pd \in S_i$. Since there are $\pi_c(k)$ prime numbers in $\{2^{4k+1}+1, \dots, 2^{k(c/2-2)}\}$ and we have $\pi_c(k) > k2^{2k}$, by the pigeonhole principle there exists $i < k$ such that $|X_i| > 2^{2k}$. Let us fix this index i.

For every $p \in X_i$, since $pd \in S_i$, there exists a closed walk $\mathcal{C}^{(p)}$ of length pd in $\mathcal{G}_i$. Since $|X_i| > 2^{2k}$, and since $\mathcal{G}_i$ has 2^{2i} vertices and $i < k$, again by the pigeonhole principle there exist $p, q \in X_i$ such that $\mathcal{C}^{(p)}$ and $\mathcal{C}^{(q)}$ have a vertex in common. Since $\mathcal{C}^{(p)}$ has length pd, $\mathcal{C}^{(q)}$ has length qd, and these two cycles have a vertex in common, for every $a, b \in \mathbb{N}$, there exists a closed walk of length $apd + bqd$ in $\mathcal{G}_i$ (obtained by starting from a vertex $u \in \mathcal{C}^{(p)} \cap \mathcal{C}^{(q)}$, taking a times $\mathcal{C}^{(p)}$ and then b times $\mathcal{C}^{(q)}$). Thus, for every $a, b \in \mathbb{N}$, $apd + bqd \in S_i$.

Finally, since $\gcd(pd, qd) = d$ (because p and q are two distinct prime numbers), since $pd, qd \leqslant \sqrt{n_k}$, and since n_k is divisible by d by Claim 29, we can apply Lemma 26 which states that there exists $a, b \in \mathbb{N}$ such that $apd + bpd = n_k$. So $n_k \in S_i$, which a contradiction, because by assumption $n_k \in S_k \setminus \bigcup_{1 \leqslant i < k} S_i$. This concludes the proof of Theorem 7.

6 A property with optimal size $\Theta(\log n)$ in cycles

Let us finish this short version of the paper with a property that can be certified with $O(\log n)$ bits but not with constant-size certificates, to show that the gap stated in Theorem 7 is optimal. The proof is in Appendix C

▶ **Proposition 31.** *Certifying that the length of a cycle is not a power of* 2 *can be done with certificates of size* $O(\log n)$ *but not with certificates of size* $O(1)$.

References

1 Karine Altisen, Stéphane Devismes, Swan Dubois, and Franck Petit. *Introduction to Distributed Self-Stabilizing Algorithms*. Morgan & Claypool Publishers, 2019. doi:10.2200/S00908ED1V01Y201903DCT015.

2 James Aspnes and Eric Ruppert. An introduction to population protocols. *Bull. EATCS*, 93:98–117, 2007.

3 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*, pages 262–271. ACM, 2019. doi:10.1145/3293611.3331606.

4 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. Efficient classification of locally checkable problems in regular trees. In Christian

Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022*, volume 246 of *LIPIcs*, pages 8:1–8:19, 2022. doi:10.4230/LIPICS.DISC.2022.8.

5 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. *Distributed Comput.*, 36(3):277–311, 2023. doi:10.1007/S00446-022-00435-9.

6 Alkida Balliu, Sebastian Brandt, Dennis Olivetti, and Jukka Suomela. Almost global problems in the LOCAL model. *Distributed Comput.*, 34(4):259–281, 2021. doi:10.1007/S00446-020-00375-2.

7 Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1307–1318. ACM, 2018. doi:10.1145/3188745.3188860.

8 Dan Alden Baterisna and Yi-Jun Chang. Optimal local certification on graphs of bounded pathwidth. *CoRR*, abs/2502.00676, 2025. doi:10.48550/arXiv.2502.00676.

9 Lélia Blin, Swan Dubois, and Laurent Feuilloley. Silent MST approximation for tiny memory. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020*, volume 12514, pages 118–132. Springer, 2020. doi:10.1007/978-3-030-64348-5_10.

10 Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, SSS 2014*, volume 8756, pages 18–32, 2014. doi:10.1007/978-3-319-11764-5_2.

11 Nicolas Bousquet, Linda Cook, Laurent Feuilloley, Théo Pierron, and Sébastien Zeitoun. Local certification of forbidden subgraphs. *CoRR*, abs/2402.12148, 2024. doi:10.48550/arXiv.2402.12148.

12 Nicolas Bousquet, Louis Esperet, Laurent Feuilloley, and Sébastien Zeitoun. Renaming in distributed certification. *CoRR*, abs/2409.15404, 2024. doi:10.48550/arXiv.2409.15404.

13 Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun. Local certification of local properties: Tight bounds, trade-offs and new parameters. In *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024*, volume 289 of *LIPIcs*, pages 21:1–21:18, 2024. doi:10.4230/LIPICS.STACS.2024.21.

14 Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun. Complexity landscape for local certification. *CoRR*, abs/2505.20915, 2025. doi:10.48550/arXiv.2505.20915.

15 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed lovász local lemma. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 479–488. ACM, 2016. doi:10.1145/2897518.2897570.

16 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemyslaw Uznanski. LCL problems on grids. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017*, pages 101–110. ACM, 2017. doi:10.1145/3087801.3087833.

17 Alfred Brauer. On a problem of partitions. *American Journal of Mathematics*, 64(1):299–312, 1942.

18 Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theor. Comput. Sci.*, 811:112–124, 2020. doi:10.1016/J.TCS.2018.08.020.

19 Yi-Jun Chang. The complexity landscape of distributed locally checkable problems on trees. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020*, volume 179 of *LIPIcs*, pages 18:1–18:17, 2020. doi:10.4230/LIPICS.DISC.2020.18.

20 Yi-Jun Chang. The distributed complexity of locally checkable labeling problems beyond paths and trees. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024,*

January 30 to February 2, 2024, Berkeley, CA, USA, volume 287 of *LIPIcs*, pages 26:1–26:25, 2024. doi:10.4230/LIPICS.ITCS.2024.26.

21 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM J. Comput.*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

22 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM J. Comput.*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

23 Yi-Jun Chang, Jan Studený, and Jukka Suomela. Distributed graph problems through an automata-theoretic lens. *Theor. Comput. Sci.*, 951:113710, 2023. doi:10.1016/J.TCS.2023.113710.

24 Linda Cook, Eun Jung Kim, and Tomás Masarík. A tight meta-theorem for LOCAL certification of mso_2 properties within bounded treewidth graphs. *CoRR*, abs/2503.19671, 2025. doi:10.48550/arXiv.2503.19671.

25 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000. URL: http://www.cs.bgu.ac.il/%7Edolev/book/book.html.

26 Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 137–146. ACM, 2013. doi:10.1145/2484239.2484244.

27 Laurent Feuilloley. Introduction to local certification. *Discret. Math. Theor. Comput. Sci.*, 23(3), 2021. doi:10.46298/DMTCS.6280.

28 Laurent Feuilloley, Nicolas Bousquet, and Théo Pierron. What can be certified compactly? compact local certification of MSO properties in tree-like graphs. In *PODC '22: ACM Symposium on Principles of Distributed Computing*, pages 131–140. ACM, 2022. doi:10.1145/3519270.3538416.

29 Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bull. EATCS*, 119, 2016. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/411.

30 Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *ACM Symposium on Principles of Distributed Computing*, pages 157–165. ACM, 2013. doi:10.1145/2484239.2484264.

31 Pierre Fraigniaud, Magnús M. Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *Principles of Distributed Systems, 16th International Conference, OPODIS 2012*, volume 7702, pages 224–238. Springer, 2012. doi:10.1007/978-3-642-35476-2_16.

32 Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. *Theor. Comput. Sci.*, 751:61–73, 2018. doi:10.1016/J.TCS.2017.01.011.

33 Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35:1–35:26, 2013. doi:10.1145/2499228.

34 Pierre Fraigniaud, Frédéric Mazoit, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed certification for classes of dense graphs. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023*, volume 281 of *LIPIcs*, pages 20:1–20:17, 2023. doi:10.4230/LIPICS.DISC.2023.20.

35 Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. *Algorithmica*, 86(2):585–612, 2024. doi:10.1007/S00453-023-01185-1.

36 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory Comput.*, 12(1):1–33, 2016. doi:10.4086/TOC.2016.V012A019.

37 Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers*. Oxford university press, 1979.

38 Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

39 Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, and Sergei Vassilvitskii. Massively parallel computation: Algorithms and applications. *Found. Trends Optim.*, 5(4):340–417, 2023. doi:10.1561/2400000025.

40 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010. doi:10.1007/S00446-010-0095-3.

41 Masayuki Miyamoto. Distributed complexity of p_k-freeness: Decision and certification. *CoRR*, abs/2410.20353, 2024. doi:10.48550/arXiv.2410.20353.

42 Jukka Suomela. Landscape of locality (invited talk). In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, volume 162 of *LIPIcs*, pages 2:1–2:1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.SWAT.2020.2.

A No gap above $\log n$ in general graphs with identifiers

We establish that there is no gap in the local certification complexity in general graphs with identifiers in the $\Omega(\log n)$ regime. The proof follows the same route as the proof of the $\Theta(n^2)$ bound for non-trivial automorphism in [36] .

▶ **Theorem 1.** *For general graphs with identifiers, for any non-decreasing function $f(n)$ in $\Omega(\log n)$ and $O(n^2)$, there exists a property that can be certified with $O(f(n))$ bits, but not in $o(f(n))$ bits.*

Since the proof is a rather direct adaptation of the one of [36] and is not central in this paper, we use a more sketchy style than for our other proofs.

Proof. Fix some function f in $\Omega(\log n)$ and $O(n^2)$ consider the following language. A graph is in the language if it is made of two copies of a graph H on $\sqrt{f(n)}$ nodes,where an arbitrary node is linked to its copy by a path of length $n-2\sqrt{f(n)}$. A certification for this language is the following. On a correct instance, every node is given the following pieces of information.

1. Its part of the certification of the size n of the graph, via a spanning tree (see [27]).
2. The adjacency matrix of H.
3. The identifier assignment restricted to the copies of H.
4. Its parts of two spanning trees, pointing to the two nodes that belong both to a copy of H and to the path.

Every node checks the following.

1. Item 1 and 4 above are consistent (again see [27]).
2. If its identifier appears in the list of Item 3, it checks that its neighborhood in the graph is consistent with the neighborhood in H as described by the certificates; except if it is the root of a spanning tree of Item 4, in which case it should have exactly one additional neighbor.
3. If its identifier does not appear in the certificates it should have degree 2.

The correctness is follows from [36], where the same scheme is used except that the central path has constant length (and therefore the identifiers of the nodes on the path can be given to all nodes without overhead).

The spanning tree certifying the number of nodes can be encoded in $O(\log n)$ bits, the adjacency matrices use $O\left(\left(\sqrt{f(n)}\right)^2\right) = O(f(n))$ bits, and the identifier assignment uses $2\sqrt{f(n)}\log n$ bits. Hence in total $O(f(n))$ bits in the regime of the theorem.

Again the lower bound is very similar to the one of [36]. Basically, if we were to use $o(f(n))$ bits, by pigeon-hole principle, there would be two different correct instances of the same size n, for which the same certificates would be used on one edge of the path. Then

we could consider the graph where we take the right part from one instance and the left part from the other (with their accepting certificates), gluing on the edge with identical certificates. This new graph would be accepted, but it is not in our language since the graphs at the end of the path are different. A contradiction. ◀

B Missing proofs for the gap between $O(1)$ and $\Theta(\log n)$ in cycles

In this appendix section, we restate and prove the claim of Section 5 for which the proof was missing.

▷ Claim 27. For every integer $n \geqslant 3$, $n \in S_k$ if and only there exists a closed walk of length n in $\mathcal{G}_k$.

Proof. If $n \in S_k$, there exists an assignment of certificates $c_1, \ldots, c_n$ to the vertices of a n-vertex cycle such that every vertex accepts. For every $i \in \{1, \ldots, n\}$, since the vertex with certificate c_i accepts and has two neighbors with certificates c_{i-1} and c_{i+1} (where $i-1$ and $i+1$ are taken modulo n), by definition there is an edge in $\mathcal{G}_k$ from (c_{i-1}, c_i) to (c_i, c_{i+1}). So this gives an closed walk of length n in $\mathcal{G}_k$. Conversely, if there exists a closed walk $(c_1, c_2), (c_2, c_3), \ldots, (c_n, c_1)$ in $\mathcal{G}_k$, by definition all the vertices of the cycle of length n with certificates $c_1, \ldots, c_n$ accept, so $n \in S_k$. ◁

▷ Claim 28. We have $n_k \geqslant 2^{ck}$.

Proof. By definition of $s(n_k)$, we have $n_k \in S_{s(n_k)}$. Since $n_k \in S_k \setminus \bigcup_{1 \leqslant i < k} S_i$, it follows that $s(n_k) \geqslant k$. Moreover, by assumption, $s(n_k) \leqslant \left\lfloor \frac{\log n_k}{c} \right\rfloor$. So $n_k \geqslant 2^{ck}$. ◁

▷ Claim 29. Let $\mathcal{C}$ be a closed walk in $\mathcal{G}'_k$, and let ℓ be its length. There exists $b_1, \ldots, b_t \in \mathbb{N}$ such that $\ell = \sum_{i=1}^{t} b_i \ell_i$. Thus, d divides ℓ. In particular, d divides n_k.

Proof. Let us prove this result by induction on the length of $\mathcal{C}$. The base case includes all elementary cycles: if $\mathcal{C}$ is an elementary cycle, there exists $j \in \{1, \ldots, t\}$ such that $\ell = \ell_j$, so the result is trivially true. Assume now that $\mathcal{C}$ is not an elementary cycle, and consider the shortest closed subwalk $\mathcal{C}'$ of $\mathcal{C}$. Then, $\mathcal{C}'$ is an elementary cycle (otherwise it would not be the shortest subwalk of $\mathcal{C}$), so $\mathcal{C}' \in \{\mathcal{C}_1, \ldots, \mathcal{C}_t\}$ and its length is equal to ℓ_j for some $j \in \{1, \ldots, t\}$. Let us denote by $\mathcal{C} \setminus \mathcal{C}'$ the closed walk obtained by removing from $\mathcal{C}$ the steps of $\mathcal{C}'$. The length of $\mathcal{C} \setminus \mathcal{C}'$ is $\ell - \ell_j$. Finally, apply the induction hypothesis to the closed walk $\mathcal{C} \setminus \mathcal{C}'$, to obtain integers $b_1, \ldots, b_t \in \mathbb{N}$ such that $\ell - \ell_j = \sum_{i=1}^{t} b_i \ell_i$. The result follows. ◁

▷ Claim 30. Let $m \in \mathbb{N}$ be such that d divides m, and $m \geqslant 2^{4k+1}$. Then, there exists a closed walk in $\mathcal{G}'_k$ of length m. Thus, $m \in S_k$.

Proof. First, we construct greedily a closed walk $\mathcal{C}_0$ in $\mathcal{G}'_k$ that passes through all the vertices of $\mathcal{G}'_k$ (it exists, because $\mathcal{G}'_k$ is strongly connected). For every $u, v \in V(\mathcal{G}'_k)$, the shortest directed path from u to v has length at most the number of vertices of $\mathcal{G}_k$ which is 2^{2k}. Thus, there exists a closed walk $\mathcal{C}_0$ of length $\ell_0 \leqslant (2^{2k})^2 = 2^{4k}$ that passes through all the vertices. By Claim 29, d divides ℓ_0, so d divides $m - \ell_0$. Furthermore, $m - \ell_0 \geqslant 2^{4k}$. Since we have $\max_{1 \leqslant i \leqslant t} \ell_i \leqslant 2^{2k}$, we can apply Lemma 26 to get the existence of integers $a_1, \ldots, a_t \in \mathbb{N}$ such that $m - \ell_0 = \sum_{i=1}^{t} a_i \ell_i$. Finally, to construct a closed walk in $\mathcal{G}'_k$ of length $m = \ell_0 + \sum_{i=1}^{t} a_i \ell_i$, we attach a_i times the elementary cycle $\mathcal{C}_i$ to the closed walk $\mathcal{C}_0$ for every $i \in \{1, \ldots, t\}$ (this is possible, because $\mathcal{C}_0$ passes through all the vertices). By Claim 27, we have $m \in S_k$. ◁

C Proof of a property with optimal size $\Theta(\log n)$ in cycles

Let us restate the result and prove it.

▶ **Proposition 32.** *Certifying that the length of a cycle is not a power of* 2 *can be done with certificates of size* $O(\log n)$ *but not with certificates of size* $O(1)$.

Proof. Let C be a cycle of length n, let $u \in V(C)$ and let P be the path obtained from C by deleting one edge adjacent to u. To certify that $n \notin \{2^k, k \in \mathbb{N}\}$, the prover writes in the certificate of every vertex $v \in V(C)$ the tuple (d, i) where $d \geqslant 3$ is an odd integer that divides n, and i is the distance from u to v in P modulo d. Every vertex checks that, if its certificate is (d, i) then its two neighbors have certificates $(d, i-1 \mod d)$ and $(d, i+1 \mod d)$, and that d is indeed odd. Such a certificate has size $O(\log n)$. This scheme is correct: indeed, if all the vertices accept, the length of the cycle should be divisible by d (and conversely, with the certificates described above, all the vertices will accept if the cycle has length divisible by d).

Now, assume by contradiction that certifying that the length n of a cycle is not a power of 2 can be done with certificates of constant size k (or equivalently, that 2^k distinct certificates are sufficient). Let p be an odd prime number such that $p > 2^k$. Let us consider an assignment of certificates to the vertices of a cycle C of length p such that all the vertices accept. Let us number the vertices of C in clockwise order starting from an arbitrary vertex, and for every $i \in \{0, \dots, n-1\}$, let u_i be the i-th vertex in this numbering, and c_i be its certificate. By the pigeonhole principle, there exists $0 \leqslant i < j \leqslant n-1$ such that $(c_i, c_{i+1}) = (c_j, c_{j+1})$ (where $j+1$ is taken modulo n). If $j = i+1$, then a vertex with certificate c_i accepts with two neighbors having certificate c_i. Thus, in this case, any cycle is accepted (by giving the certificate c_i to all the vertices) which is a contradiction. Else, let $\ell_1 := j - i$ and $\ell_2 := p - j + i$. We have $\ell_1, \ell_2 \in \{1, \dots, p-1\}$ and $\ell_1 + \ell_2 = p$. Since p is prime, we get $\gcd(\ell_1, \ell_2) = 1$. Moreover, for any $a_1, a_2 \in \mathbb{N}$, a cycle of size $a_1\ell_1 + a_2\ell_2$ is accepted. Indeed, by cutting such a cycle in a_1 portions of length ℓ_1 and a_2 portions of length ℓ_2, giving the certificates $c_i, \dots, c_{j-1}$ to the vertices in portions of size ℓ_1 and $c_j, \dots, c_{n-1}, c_0, \dots, c_{i-1}$ to the vertices in portions of length ℓ_2, all the vertices accept because they have the same view as a vertex which accepts in C. Using Lemma 26, all the the cycles of length $m \geqslant p^2$ are accepted, which is a contradiction because the cycles whose length is a power of 2 greater than p^2 should not be accepted. ◀

Robust Predicate and Function Computation in Continuous Chemical Reaction Networks

Kim Calabrese
University of California–Davis, CA, USA

David Doty
University of California–Davis, CA, USA

Mina Latifi
University of California–Davis, CA, USA

Abstract

We initiate the study of "rate-constant-independent" computation of Boolean predicates (decision problems) and numerical functions in the continuous model of chemical reaction networks (CRNs), which model the amount of a chemical species as a nonnegative, real-valued *concentration*, representing an average count per unit volume. Real-valued numerical functions have previously been studied [20], finding that exactly the continuous, piecewise rational linear (meaning linear with rational slopes) functions $f : \mathbb{R}^k_{>0} \to \mathbb{R}_{>0}$ can be computed *stably* (a.k.a., *rate-independently*), meaning roughly that the CRN gets the answer correct no matter the rate at which reactions occur. For example the reactions $X_1 \to Y$ and $X_2 + Y \to \emptyset$, starting with inputs $X_1 \geq X_2$, converge to output Y having concentration equal to the initial difference of inputs $X_1 - X_2$, no matter the relative rate at which each reaction proceeds.

We first show that, contrary to the case of real-valued functions, continuous CRNs are severely limited in the Boolean predicates they can stably decide, reporting a yes/no answer based only on which inputs are 0 or positive, but not on the exact positive value of any input.

This limitation motivates a slightly relaxed notion of rate-independent computation in CRNs that we call *robust computation*. The standard mass-action rate model is used, in which each reaction (e.g., $A + B \xrightarrow{k} C$) is assigned a *rate* ($A \cdot B \cdot k$ in this example) equal to the product of its reactant concentrations and its *rate constant* k. We say the computation is correct in this model if it converges to the correct output for *any* positive choice of rate constants. This adversary is weaker than the adversary defining stable computation, the latter being able to run reactions at rates that are not those of mass-action for any choice of rate constants (e.g., the stable adversary may deactivate a reaction temporarily, even if all reactants are positive).

We show that CRNs can robustly decide every predicate that is a finite Boolean combination of *threshold predicates*, where a threshold predicate is defined by taking a rational weighted sum of the inputs $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$ and comparing to a constant, answering the question "Is $\sum_{i=1}^k w_i \cdot \mathbf{x}(i) > h$?", for rational weights w_i and real threshold h. Turning to function computation, we show that CRNs can robustly compute any piecewise affine function with rational coefficients, where threshold predicates determine which affine piece to evaluate for a given input $\mathbf{x}$.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases chemical reaction networks, analog computation, mean-field limit

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.19

Related Version *Full Version*: https://arxiv.org/abs/2506.06590 [18]

Funding *Kim Calabrese*: NSF awards 2211793, 1900931.
David Doty: NSF awards 2211793, 1900931 and DoE award DE-SC0024467.
Mina Latifi: NSF awards 2211793, 1900931 and DoE award DE-SC0024467.



© Kim Calabrese, David Doty, and Mina Latifi;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

A *chemical reaction network* (CRN) is a model of reactions among abstract chemical species, such as the reaction $X + Y \to Z$, which indicates that if an X and a Y molecule collide, they can stick together to form a dimer called Z. Since the 19th century [29, 39], this model has been used to describe and predict the behavior of naturally occurring chemicals. It was not until the 21st century, however, that the model was repurposed as a *programming* language [35] for describing the desired behavior of synthetically engineering chemicals such as DNA strand displacement systems, both in theory [36] and practice [22, 38].

CRNs are related to a distributed computing model known as *population protocols* [5], describing anonymous finite-state agents (molecules) that interact (react) asynchronously in pairs (bimolecular reactions), changing state (chemical species) in response. Formally, a population protocol is a CRN in which each reaction has exactly two reactants (inputs) and two products (outputs), with unit *rate constants* (constant multiplier on a reaction rate; see below for definitions.) A population protocol is defined by a transition function $\delta : \Lambda \times \Lambda \to \Lambda \times \Lambda$, where for $A, B, C, D \in \Lambda$, $\delta(A, B) = (C, D)$ is the same as saying the CRN has a reaction $A + B \to C + D$. Although population protocols are a special case of CRNs, nevertheless many computations achievable with more general CRNs can be simulated exactly by population protocols, and conversely most impossibility results on population protocols also apply to the more general CRN model.

Discrete (a.k.a., *stochastic*) CRNs model the amount of each species as a nonnegative integer representing its exact molecular count [28]. In this model, the state of a CRN is given by a continuous-time Markov chain [28], where the probability of a reaction such as $2A + B \xrightarrow{k} C$ occurring is proportional to the counts of A and B, as well as the positive *rate constant* $k \in \mathbb{R}$. Furthermore, reactions can only occur when at least two molecule of A and one molecule of B is present in the configuration. When the number of molecules is very large (a typical DNA nanotechnology experiment may involve over a trillion molecules in a 50 μL test tube), CRNs are well-approximated by what is sometimes called the *mean-field limit*, the *continuous mass-action* model, and very commonly "*the deterministic model*" to contrast with the stochastic nature of the discrete model. In the continuous model, each species has a nonnegative, real-valued *concentration* indicating its average count per unit volume.[1] In this setting a reaction such as $A + B \xrightarrow{k} 2C$ indicates that some real-valued amount of reactants A and B are converted into product species C, e.g., the reaction could turn 1/3 units of A, and the same amount of B, into 2/3 units of C. That is, in the continuous model of CRNs, a reaction encodes the *ratio* of reactants that are turned into products. This is a slightly different interpretation of a reaction than in the discrete model, where the same reaction would be interpreted as "exactly one copy of species A reacts with exactly one copy of species B to produce exactly two copies of species C". While the discrete CRN model has a state that changes at discrete steps, the state of a continuous CRN experiences continuous time evolution. Each reaction has a *rate* equal to the product of the reactant concentrations and rate constant k - $k \cdot A \cdot B$ in this example. The rates define a system of polynomial ordinary differential equations (ODEs) whose unique solution trajectory[2] indicates how species concentrations change over time. The function $S(t)$ which describes

[1] This continuous mean-field limit has been explored within population protocols as well [10, 15, 16].

[2] Polynomials are locally Lipschitz, so by the Picard-Lindelöf Theorem, the ODEs have a *unique* solution; hence the term "the deterministic model".

the concentration of species S at time t is defined by an ODE for $S(t)$, where a reaction in which S appears as a product contributes a positive term to dS/dt and a negative term when S is a reactant; see Section 2.2 for a formal definition.[3]

1.1 Computation with CRNs: Related work

The discrete, integer-valued CRN model has been far more extensively studied for its computational abilities than the continuous model, so we briefly review what is known in this setting. The two most studied types of computation are Boolean-valued predicates $\phi : \mathbb{N}^k \to \{0,1\}$ [6, 9] and numeric functions $f : \mathbb{N}^k \to \mathbb{N}$ [21, 25]. We designate special *input species* $X_1, \ldots, X_k$ whose initial counts represent the inputs to ϕ or f. These may be the only species present initially ("leaderless" CRNs [5, 9, 25]), or there may be additional species present initially with counts independent of the inputs ("leader-driven" [7, 21]). For a function $f : \mathbb{N}^k \to \mathbb{N}$, the count of a special "output" species Y represents the output of f. For a predicate $\phi : \mathbb{N}^k \to \{0,1\}$, some species are designated as "yes" voters and some as "no" voters; the CRN's output is undefined if voters for both outputs (or neither) are present, otherwise the output is yes or no depending on which type of voter is present.

Much study has been devoted to *stable* (a.k.a., *rate-independent*) computation, which means intuitively that the CRN generates the correct output no matter in what order reactions proceed. In the discrete stochastic model, rates influence the probability of choosing among several competing reactions to be the next reaction to occur, so rate-independent computation means essentially that, despite the inherent stochasticity of the model, nevertheless the correct output is generated with probability 1.[4] Experimentally, rate constants are often difficult to control precisely, sometimes with a 10-fold difference in rate constants intended to be equal [38], whereas reaction stoichiometry (i.e., how many of each reactant is consumed, and how many of each product is produced) is naturally digital and simpler to engineer exactly. Furthermore there may be violations of the assumptions justifying typical rate laws (for example the solution may not be well-mixed) leading to actual reaction rates deviating from predictions. Whether leaderless or not, exactly the *semilinear* predicates [6] and functions [21, 25] can be stably computed by discrete CRNs. See [6, 9, 21, 25] for formal definitions of these concepts, which are not required to understand this paper.

In this paper, we study the (naturally deterministic) continuous model, we focus on the "rate-independent" characterization of stable computation rather than on probability. Intuitively we want to capture the idea that the CRN generates the correct output "no matter the reaction rates". In contrast, if reaction rates are undisturbed and rate constants can be controlled, then continuous CRNs are Turing universal [27], a consequence of the surprising computational power of polynomial ODEs [17].

Prior work has completely characterized the real-valued functions $f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ stably computable by continuous CRNs [20]. This definition of stable computation is based on a very general notion of reachability, defined formally in Section B.1. A function f is stably computable by a continuous CRN if and only if it is piecewise rational linear (meaning a finite union of functions that are linear with rational coefficients, e.g., $\frac{4}{3}x_1 - 5x_2$), and *positive-continuous*, meaning intuitively that discontinuities can occur only when some input x_i goes from 0 to positive (Definition B.9); for example the function $f(x_1, x_2) = x_1 + x_2$ if $x_2 > 0$, and $f(x_1, x_2) = 3x_1$ if $x_2 = 0$, is positive-continuous but not continuous.

[3] We will often write the time derivative of S, dS/dt as S'.
[4] There are caveats to this, unnecessary to understand this paper; see [23] for a thorough discussion.

1.2 Our results

An open question from [20] concerns stable computation of real-valued decision problems, a.k.a., Boolean predicates $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$. In contrast to functions, we show that the stably decidable predicates are severely limited: exactly the *detection* predicates can be stably decided (Theorem B.2), those that depend only on whether certain inputs are 0 or positive (Definition B.1), but not on their exact positive value. This limitation prompts us to relax the notion of stable computation as suggested by another open question in [20]. We call the concept *robust computation*, which uses mass-action rate laws (see Section 2.2), limited compared to the adversarial rates used in stable computation, but requires the CRN to converge to the correct output for *every* choice of positive rate constants (Definitions 2.6 and 2.7).

Our first main result (Theorem 3.7) shows that every finite Boolean combination of *threshold predicates* (Definition 2.2) is robustly decidable, where a *threshold predicate* $\phi(\mathbf{x}) = 1$ if and only if $\sum_{i=1}^{k} w_i \cdot \mathbf{x}(i) > h$ for rational constants $w_i \in \mathbb{Q}$ and real constant $h \in \mathbb{R}$. Intuitively, there are a finite number of hyperplanes, which cut $\mathbb{R}^k_{\geq 0}$ into a finite number of regions, and $\phi(\mathbf{x})$ depends only on which region $\mathbf{x}$ is in. A famous example of a threshold predicate (which incidentally is not a detection predicate) is *majority* [1–4, 7, 8, 11–14, 24, 26, 31–33, 37] (see Lemma 3.3).

Our second main result (Theorem 3.10) shows that every *threshold-piecewise rational floor-affine function* (Definition 2.3) can be stably computed, which are finite unions of "floor-affine" components $f_i : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$, where a threshold predicate $\phi(\mathbf{x})$ indicates whether f_i is the correct affine function to use for input $\mathbf{x}$ (see Section 3.2). *Floor*-affine means negative outputs are replaced with 0 (Definition 2.1, necessary since concentrations are nonnegative), i.e., $f_i(\mathbf{x}) = \max(0, g(\mathbf{x}))$ for some affine $g : \mathbb{R}^k_{\geq 0} \to \mathbb{R}$. Compared to stable function computation [20], such functions can be discontinuous even within the strictly positive orthant; e.g., $f(x_1, x_2) = x_1 + x_2 + 2$ if $x_1 < x_2$ and $\max(0, x_1/3 - x_2)$ if $x_1 \geq x_2$.

Threshold predicates and affine functions allow a constant "offset", unlike linear functions studied in [20]. This is because unlike [20], we allow a notion similar to "leaders" in discrete distributed computing models, called *initial context*: some non-input species may be present initially, but their initial concentrations do not depend on the input values. Such initial context can be used to implement these constant offsets.[5] We believe our constructions extend naturally to the leaderless setting, and would limit threshold predicates to comparing to $h = 0$ instead of arbitrary constant $h \in \mathbb{R}$, and limit functions to piecewise *linear* (rather than affine), but we have not explored this in detail. Note that some functions are computable with specific rate constants, but are neither stably nor robustly computable; for example $f(x) = 3.5x^2$ is computable by the reactions $2X \xrightarrow{3.5} 2X + Y$ and $Y \xrightarrow{1} \emptyset$.

The high-level goals of stable and robust computation are the same, which is to formalize a notion of computation in an adversarial environment where reaction rates may deviate from standard models. However, robust computation, being based on ODEs in mass-action kinetics, requires vastly different (and typically more sophisticated) techniques to reason about. One goal of this paper is to begin establishing general techniques to prove correctness of such systems. For example, Lemmas 3.1 and 3.2 in Section 3.1 are general lemmas that we use repeatedly to reason "modularly" when the output of an "upstream" CRN U influences a "downstream" CRN D in one direction only, i.e., D does not influence U.

[5] As noted in [20, Section 6.2], allowing initial context in stable computation leads to replacing "linear" with "affine" in the characterization.

2 Preliminaries

Let $\mathbb{N}$ denote the nonnegative integers, $\mathbb{Q}$ the rationals, and $\mathbb{R}$ the reals. For any set $A \subseteq \mathbb{R}$, $A_{\geq 0} = A \cap [0, \infty)$ and $A_{>0} = A \cap (0, \infty)$. Given a finite set F and a set S, let S^F denote the set of functions $\mathbf{c} : F \to S$. In the case of $S = \mathbb{R}$ (resp., $\mathbb{N}$), we view $\mathbf{c}$ equivalently as a real-valued (resp., integer-valued) vector indexed by elements of F. Given $a \in F$, we write $\mathbf{c}(a)$, to denote the real number indexed by a. The notation $\mathbb{R}^F_{\geq 0}$ is defined similarly for nonnegative real vectors. For a function of time $A : \mathbb{R}_{\geq 0} \to \mathbb{R}$ and $c \in \mathbb{R} \cup \{\infty\}$, we write $A \to c$ to denote $\lim_{t\to\infty} A(t) = c$; most frequently, A will be a chemical species concentration, or some function thereof.

▶ **Definition 2.1.** *$f : \mathbb{R}^k \to \mathbb{R}$ is* rational affine *if for some rational $w_1, \ldots, w_k \in \mathbb{Q}$ and real $h \in \mathbb{R}$, for all $\mathbf{x} \in \mathbb{R}^k$, $f(\mathbf{x}) = h + \sum_{i=1}^k w_i \cdot \mathbf{x}(i)$.[6] If $h = 0$ we say f is* rational linear. *f is* rational floor-affine *if $f(\mathbf{x}) = \max(0, g(\mathbf{x}))$ for some rational affine g.*[7]

For example, the function $f_1(x, y, z) = \sqrt{2} + 2x + \frac{2}{3}y - \frac{3}{7}z$ is rational affine while the function $f_2(x, y, z) = 1 + \sqrt{2}x + \sqrt{3}y + 2z$ is not, although still affine.

The next two definitions capture the class of predicates and functions we show are robustly computable by continuous CRNs in our main results, Theorems 3.7 and 3.10.

▶ **Definition 2.2.** *$\phi : \mathbb{R}^k \to \{0, 1\}$ is a* threshold predicate *with rational* weights $w_1, \ldots, w_k \in \mathbb{Q}$ *and real* threshold $h \in \mathbb{R}$ *if, for all $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$, $\phi(\mathbf{x}) = 1 \iff \sum_{i=1}^k w_i \cdot \mathbf{x}(i) > h$. We say ϕ is a* multi-threshold predicate *if it is a finite Boolean combination of threshold predicates.*

For example, the predicate $\phi(x, y, z) = 1 \iff x + 2y + 3z > \pi$ is a threshold predicate with rational weights $1, 2$ and 3 and threshold π. If ϕ_1, ϕ_2 and ϕ_3 are each threshold predicates, then the predicate ψ defined as $\psi(\mathbf{x}) = (\phi_1(\mathbf{x}) \vee \phi_2(\mathbf{x})) \wedge \overline{\phi_3}(\mathbf{x})$ (where $\vee$ and $\wedge$ denote logical OR and AND, and $\overline{\phi_3}$ is the logical negation of ϕ_3) is a multi-threshold predicate.

▶ **Definition 2.3.** *$f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ is* threshold-piecewise rational floor-affine *if there is a finite set of rational floor-affine functions $f_1, \ldots, f_l : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$, known as the* affine components *of f, and multi-threshold predicates $\phi_1, \ldots, \phi_l : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ such that*

1. *The sets $\phi_1^{-1}(1), \ldots, \phi_l^{-1}(1)$ are a partition of $\mathbb{R}^k_{\geq 0}$, i.e., for each $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$, $\phi_i(\mathbf{x}) = 1$ for exactly one $1 \leq i \leq l$.*
2. *For each i and $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$, if $\phi_i(\mathbf{x}) = 1$ then $f(\mathbf{x}) = f_i(\mathbf{x})$, i.e., $\phi_i(\mathbf{x})$ indicates whether f_i is the correct affine component defining $f(\mathbf{x})$.*

For example, if $f_1, f_2 : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ are floor-affine functions and $\phi : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a multi-threshold predicate then the function

$$f(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) & \text{If } \phi(\mathbf{x}) = 1 \\ f_2(\mathbf{x}) & \text{If } \overline{\phi}(\mathbf{x}) = 1 \end{cases}$$

is a threshold-piecewise rational floor-affine function as both ϕ and its negation $\overline{\phi}$ are multi-threshold predicates, and for any input $\mathbf{x}$ exactly one of $\phi(\mathbf{x})$ and $\overline{\phi}(\mathbf{x})$ is 1.

[6] It would seem more natural to require h also to be rational, or conversely to allow each w_i to be real, but this distinction is relevant with CRNs: h will come from the initial concentration of some species, or its negation, whereas the w_i's will come from ratios of integer stoichiometric reaction coefficients. Similar reasoning applies to Definition 2.2.

[7] The significance of rational floor-affine functions is that CRNs can only output nonnegative concentrations; reactions such as $X_1 \to Y$ and $X_2 + Y \to \emptyset$ technically compute, not the affine function $g(x_1, x_2) = x_1 - x_2$, but the floor-affine function $f(x_1, x_2) = \max(0, g(x_1, x_2))$.

2.1 Chemical reaction networks

Throughout this paper, let Λ be a finite set of chemical *species*. Given $S \in \Lambda$ and *state* $\mathbf{c} \in \mathbb{R}^{\Lambda}_{\geq 0}$, $\mathbf{c}(S)$ is the *concentration of S in* $\mathbf{c}$. For any $\mathbf{c} \in \mathbb{R}^{\Lambda}_{\geq 0}$, let $[\mathbf{c}] = \{S \in \Lambda \mid \mathbf{c}(S) > 0\}$, the set of species *present* in $\mathbf{c}$ (a.k.a., the *support* of $\mathbf{c}$). We write $\mathbf{c} \leq \mathbf{c}'$ to denote that $\mathbf{c}(S) \leq \mathbf{c}'(S)$ for all $S \in \Lambda$. Given $\mathbf{c}, \mathbf{c}' \in \mathbb{R}^{\Lambda}_{\geq 0}$, we define the vector component-wise operations of addition $\mathbf{c} + \mathbf{c}'$, subtraction $\mathbf{c} - \mathbf{c}'$, and scalar multiplication $x\mathbf{c}$ for $x \in \mathbb{R}$.

A *reaction* over Λ is a triple $\alpha = (\mathbf{r}, \mathbf{p}, k) \in \mathbb{N}^{\Lambda} \times \mathbb{N}^{\Lambda} \times \mathbb{R}_{>0}$, such that $\mathbf{r} \neq \mathbf{p}$, specifying the stoichiometry of the *reactants* $\mathbf{r}$ and *products* $\mathbf{p}$, and the *rate constant* k. For instance, given $\Lambda = \{A, B, C, D\}$, the reaction $A + 2B \xrightarrow{6.7} A + 3C$ is the triple $((1, 2, 0, 0), (1, 0, 3, 0), 6.7)$.

A *chemical reaction network (CRN)* is a pair $\mathcal{C} = (\Lambda, R)$, where Λ is a finite set of chemical *species*, and R is a finite set of reactions over Λ. A *state* of a CRN $\mathcal{C} = (\Lambda, R)$ is a vector $\mathbf{c} \in \mathbb{R}^{\Lambda}_{\geq 0}$. Given a state $\mathbf{c}$ and reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$, we say that α is *applicable* in $\mathbf{c}$ if $[\mathbf{r}] \subseteq [\mathbf{c}]$ (i.e., $\mathbf{c}$ contains positive concentration of all of the reactants). If no reaction is applicable in state $\mathbf{c}$, we say $\mathbf{c}$ is *static*.

The next two definitions are "syntactic" preparation for stating how a CRN can compute a predicate or function; the "semantic" definitions of stable (Definitions B.5 and B.6) and robust (Definitions 2.6 and 2.7) computation will use these definitions to state under what conditions a CRN "correctly" computes. The first definition is for Boolean predicates.

▶ **Definition 2.4.** *A* chemical reaction decider *(CRD) is a tuple* $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \mathbf{i})$ *where* (Λ, R) *is a CRN,* $\Sigma \subseteq \Lambda$ *is the set of* input species*,* $\Upsilon_{\text{yes}} \subseteq \Lambda$ *is the set of* yes voters*,* $\Upsilon_{\text{no}} \subseteq \Lambda \backslash \Upsilon_{\text{yes}}$ *is the set of* no voters*, and* $\mathbf{i} \in \mathbb{R}^{\Lambda \backslash \Sigma}_{\geq 0}$ *is the* initial context*.*

Intuitively initial context $\mathbf{i}$ refers to fixed initial concentrations for non-input species, independent of the input value. A CRD's initial state for predicate input $\mathbf{x} \in \mathbb{R}^{k}_{\geq 0}$ is then $\mathbf{x} + \mathbf{i}$, where we assume some fixed ordering $X_1, \ldots, X_k$ of input species to interpret a vector $\mathbf{x} \in \mathbb{R}^{k}_{\geq 0}$ as a state $\mathbf{x} \in \mathbb{R}^{\Sigma}_{\geq 0}$. The next definition is used for computing numeric functions, identifying a special species Y whose concentration represents output:

▶ **Definition 2.5.** *A* chemical reaction computer *(*CRC*) is a tuple* $\mathcal{C} = (\Lambda, R, \Sigma, Y, \mathbf{i})$ *where* (Λ, R) *is a CRN,* $\Sigma \subseteq \Lambda$ *is the set of* input species*, and* Y *is the* output species*, and* $\mathbf{i} \in \mathbb{R}^{\Lambda \backslash \Sigma}_{\geq 0}$ *is the* initial context*.*

2.2 Robust (rate-*constant*-independent) computation

Stable computation requires a CRN to work against a very powerful adversary who can essentially set the rate of each reaction at each time arbitrarily. This means in particular that the CRN works under a variety of rate laws besides mass-action (defined below). Here we consider a weaker adversary, one that cannot control the rate law – that will be mass-action – but that *can* set the parameters of the rate law, known as *rate constants*. Crucially, these are constant with respect to *time*: the adversary can choose arbitrary positive values for these rate constants, but sets them to those values at time $t = 0$, and the rate constants stay at those values for all future $t > 0$.[8] A CRN robustly computes a function or predicate if it computes the correct output against this adversary: i.e., if the mass-action rate law converges to the correct output, no matter which positive rate constants are chosen.

[8] If we allowed the adversary to change the rate constants over time, then it could mimic the stable computation adversary by adjusting rate constants so as to target particular absolute rates at each time.

A CRN $\mathcal{C} = (\Lambda, R)$ under the *mass-action rate law* is governed by a system of polynomial ordinary differential equations (ODEs) that define functions $S(t)$ representing the concentration of species S at time t. The *rate* $\rho_t(\alpha)$ of a reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ at time t is $\rho_t(\alpha) = k \cdot \prod_{S \in \Lambda} S(t)^{\mathbf{r}(S)}$, i.e., the rate constant times each reactant concentration at time t. For example, the rate of $A + 2B \xrightarrow{4.5} C$ is $4.5 \cdot A(t) \cdot B(t)^2$. Each reaction $\alpha = (\mathbf{r}, \mathbf{p}, k)$ contributes a term $\rho_t(\alpha) \cdot (\mathbf{p}(S) - \mathbf{r}(S))$ to the ODE of each species S that is net produced or consumed; the term is α's rate $\rho_t(\alpha)$ times the net stoichiometry of S in α (positive if S is net produced by α, e.g., $S \to 3S$ net produces 2 S's since $\mathbf{p}(S) - \mathbf{r}(S) = 2$ and negative if S is net consumed, e.g., $2S \to S$ net consumes 1 S, i.e., $\mathbf{p}(S) - \mathbf{r}(S) = -1$). For example, the CRN with reactions $A + 2B \xrightarrow{k_1} 3C$ and $2C \xrightarrow{k_2} C$ corresponds to the ODEs

$$A'(t) = -k_1 A(t) B(t)^2$$
$$B'(t) = -2k_1 A(t) B(t)^2$$
$$C'(t) = 3k_1 A(t) B(t)^2 - k_2 C(t)^2$$

This can be formalized as follows. Given a CRN $\mathcal{C} = (\Lambda, R)$ let $\mathbf{A} : \mathbb{R}^{\Lambda}_{\geq 0} \to \mathbb{R}^{R}_{\geq 0}$ map each state $\mathbf{d}$ of $\mathcal{C}$ to the vector $\mathbf{A}(\mathbf{d})$ of instantaneous reaction rates in state $\mathbf{d}$, as given by the mass-action ODEs. With the example reactions above, the state of concentrations $\mathbf{d} = (1, 2, 0)$ would be mapped to the flux vector $\mathbf{A}(\mathbf{d}) = (-4k_1, -8k_1, 12k_1)$. The first coordinate was obtained by setting $A(t) = 1$, $B(t) = 2$ and $C(t) = 0$ in the ODE for A'. The second and third coordinates are obtained similarly by evaluating the ODE for B' and C' respectively. Define the $|\Lambda| \times |R|$ *stoichiometry matrix* $\mathbf{M}$ such that, for species $S \in \Lambda$ and reaction $\alpha = (\mathbf{r}, \mathbf{p}) \in R$, $\mathbf{M}(S, \alpha) = \mathbf{p}(S) - \mathbf{r}(S)$ is the net amount of S produced by α (negative if S is consumed). For example, if we have the reactions $X \to Y$ and $X + A \to 2X + 3Y$, and if the three rows correspond to A, X, and Y, in that order, then $\mathbf{M} = \begin{pmatrix} 0 & -1 \\ -1 & 1 \\ 1 & 3 \end{pmatrix}$. The vector $\mathbf{M} \cdot \mathbf{A}(\mathbf{d})$ gives the rate at which each species concentration is changing in state $\mathbf{d}$. Given an initial state $\mathbf{c}$, the *mass-action trajectory* $\boldsymbol{\tau} : [0, t_{\max}) \to \mathbb{R}^{\Lambda}_{\geq 0}$ *starting at* $\mathbf{c}$, is the solution to the initial value problem

$$\frac{d\boldsymbol{\tau}}{dt} = \mathbf{M} \cdot \mathbf{A}(\boldsymbol{\tau}(t)), \ \boldsymbol{\tau}(0) = \mathbf{c} \tag{1}$$

where $t_{\max} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$.[9] That is, $\boldsymbol{\tau}(t) = (S_1(t), S_2(t), \ldots, S_{|\Lambda|}(t))$ is the vector whose ith component is the concentration of species $S_i \in \Lambda$ at time t. While some CRNs induce ODEs with solutions that do not exist for all time, mass-action ODEs are locally Lipschitz, implying that a unique solution to (1) always exists on some interval. We now define what it means for CRD to decide a predicate in the mass action model with adversarial rate constants.

Intuitively, a CRD robustly decides a predicate ϕ if, for any input $\mathbf{x}$ and any choice of rate constants for the reactions, the induced ODEs (defined by mass action) cause yes voters to remain positive as $t \to \infty$ while no voters converge to zero if $\phi(\mathbf{x}) = 1$ and vice-versa if $\phi(\mathbf{x}) = 0$.

▶ **Definition 2.6** (robustly decide). *Let $\phi : \mathbb{R}^{k}_{\geq 0} \to \{0, 1\}$ be a predicate. We say a CRD $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \mathbf{i})$* robustly decides *(a.k.a.,* rate-constant-independently decides*) ϕ if, for any choice of strictly positive rate constants and every $\mathbf{x} \in \mathbb{R}_{\geq 0}$, the following holds.*

[9] Although $t_{\max} = \infty$ for "typical" CRNs, there are pathological CRNs such as $2X \to 3X$, which diverge to ∞ in finite time; for instance with $X(0) = 1$ and unit rate constant, this CRN has the solution $X(t) = 1/(1 - t)$, which goes to ∞ as $t \to 1$, so $t_{\max} = 1$ for this CRN.

Let $\boldsymbol{\tau}$ be the mass-action trajectory of $\mathcal{C}$, starting at state $\mathbf{x}+\mathbf{i}$. If $\phi(\mathbf{x})=1$ (resp. 0), let $\Upsilon_{\mathrm{C}}=\Upsilon_{\mathrm{yes}}$ (resp. Υ_{no}) be the correct voters, and let $\Upsilon_{\mathrm{I}}=\Upsilon_{\mathrm{no}}$ (resp. Υ_{yes}) be the incorrect voters. Define $C(t)=\sum_{V\in\Upsilon_{\mathrm{C}}}V(t)$ be the sum of concentrations of correct voters. Then $\liminf_{t\to\infty}C(t)>0$ and $\lim_{t\to\infty}I(t)=0$ for all $I\in\Upsilon_{\mathrm{I}}$.

In other words, the CRD deciding $\phi(\mathbf{x})$ starts with input species concentrations defined by $\mathbf{x}$, and other initially present species indicated by $\mathbf{i}$ (whose initial concentrations are the same for all inputs $\mathbf{x}$). The CRD proceeds by mass-action dynamics as defined above, and converges to a state with only correct voters present. Since concentrations are nonnegative, requiring each individual incorrect voter to converge to 0 is equivalent to requiring their sum to converge to 0, whereas we do not require any individual correct voter to stay bounded above 0, only the sum of correct voters,[10]

We now define what it means for a CRC to robustly compute a real-valued function.

▶ **Definition 2.7** (robustly compute). *Let $f:\mathbb{R}^k_{\geq 0}\to\mathbb{R}_{\geq 0}$. We say a CRC $\mathcal{C}=(\Lambda,R,\Sigma,Y,\mathbf{i})$* robustly computes *(a.k.a.,* rate-constant-independently computes*) f if, for any choice of strictly positive rate constants and every $\mathbf{x}\in\mathbb{R}_{\geq 0}$, the component $Y(t)$ of $\mathcal{C}$'s mass action trajectory starting from $\mathbf{i}+\mathbf{x}$ satisfies $\lim_{t\to\infty}Y(t)=f(\mathbf{x})$.*

The full definition of stable computation is given in Section B, based on a formal definition of reachability in continuous CRNs. The definition intuitively says a CRN can reach from state $\mathbf{x}$ to $\mathbf{c}$ if one can run some reactions starting at $\mathbf{x}$ and reach to $\mathbf{c}$, without ever running a reaction when one of its reactants is 0. We say that a CRC *stably computes* a function f (or stably decides a predicate ϕ) if, starting from initial state $\mathbf{x}$ encoding the input, for any state $\mathbf{c}$ reachable from $\mathbf{x}$, there is a "correct" state $\mathbf{o}$ reachable from $\mathbf{c}$ (correct meaning the output in $\mathbf{o}$ equals $f(\mathbf{x})$ or $\phi(\mathbf{x})$), that is also *stable*, meaning that every state $\mathbf{o}'$ reachable from $\mathbf{o}$ has the same output as $\mathbf{o}$.

The next definition connects stable and robust computation for some specially structured CRNs. Intuitively, a CRN is *feedforward* if there is an ordering of species so that every reaction producing a species consumes another species earlier in the ordering; formally:

▶ **Definition 2.8.** *A CRN $\mathcal{C}=(\Lambda,R)$ is* feedforward *if Λ can be ordered $\Lambda=\{S_1,S_2,\ldots,S_n\}$ so that, if for each reaction $\alpha=(\mathbf{r},\mathbf{p},k)\in R$ and S_j where $\mathbf{p}(j)>\mathbf{r}(j)$, there is $S_i\in\Lambda$ with $i<j$ such that $\mathbf{p}(i)<\mathbf{r}(i)$.*

The following was shown in [20, Corollary 4.11] (in different but equivalent terms).

▶ **Lemma 2.9.** *Each feedforward CRC stably computing function f also robustly computes f.*

Essentially the same proof shows the following.

▶ **Lemma 2.10.** *Each feedforward CRD stably deciding a predicate ϕ also robustly decides ϕ.*

3 Robust computation by continuous CRNs

3.1 Boolean combinations of threshold predicates are robustly decidable

We begin by proving a technical lemma that relates known asymptotic behavior of certain species to the desired asymptotic behavior of others that depend on them. Intuitively, we think of p and g as functions of concentration whose asymptotic behavior has already been

[10] For example two yes voters could oscillate between 0 and 1, so long as they always sum to at least 0.1.

analyzed. In particular, we think of species involved in p and g as belonging to an "upstream" CRC (or CRD) $\mathcal{C}_{\mathrm{U}}$, whose outputs are used in reactions of a "downstream" CRC (or CRD) $\mathcal{C}_{\mathrm{D}}$ that influence concentration of species F. These species influence the concentration of species F, which evolves as $f(t)$. However, f does not affect the concentrations of the species involved in p and g. Assuming that the ODE describing f is of the form $f(t)' = g(t) - p(t)f(t)$, we can use Lemmas 3.1 and 3.2 to reason about asymptotic behavior of f.

▶ **Lemma 3.1.** *Let $p, g : \mathbb{R}_{\geq 0} \to \mathbb{R}$, be differentiable functions, with $p(t) > 0$ and $g(t) \geq 0$ for all $t \in \mathbb{R}_{\geq 0}$. Let $K = \frac{g(0)+p(0)f(0)}{p(0)}$ be a constant. If $f : \mathbb{R} \to \mathbb{R}$ is differentiable and satisfies the first order linear ODE $f'(t) = g(t) - p(t)f(t)$, then for all $t \in \mathbb{R}_{\geq 0}$,*

$$f(t) \leq \frac{g(t)}{p(t)} + K \exp\left(-\int_0^t p(s)ds\right).$$

This lemma allows us to cleanly demonstrate that the concentration of particular voting species V converges to zero. In this context, the function p will represent the concentration of species whose presence causes V to be consumed, and g represents the concentration of species whose presence causes V to be produced. A common pattern in our correctness proofs will be to rearrange the mass-action ODEs for an incorrect voting species into a first order linear ODE, apply Lemma 3.1, and then argue that $g(t)$ converges to zero while $p(t)$ converges to a positive value (or converges to zero slower than g). Further, the non-negativity condition of the hypothesis is trivially satisfied as components of the mass action trajectory are always non-negative, and for some CRNs, strictly positive with suitable initial conditions. Lemma 3.1 is not applicable if $p(t)$ is not strictly positive, but in this case we can still find a bound of a similar form.

▶ **Lemma 3.2.** *Let $p, g : \mathbb{R}_{\geq 0} \to \mathbb{R}$ be differentiable functions, with $p(t) \geq 0$ and $g(t) \geq 0$ for all $t \in \mathbb{R}_{\geq 0}$. Let $K = \frac{g(0)+f(0)(p(0)+1)}{p(0)+1}$ be a constant. If $f : \mathbb{R} \to \mathbb{R}$ is differentiable and satisfies the first order linear ODE $f'(t) = g(t) - p(t)f(t)$, then for all $t \in \mathbb{R}_{\geq 0}$,*

$$f(t) \leq \frac{5g(t)}{p(t) + (t^2+1)^{-1}} + K \exp\left(-\int_0^t p(s)ds\right).$$

Proofs of Lemmas 3.1 and 3.2 are given in [18].

We begin our positive results by showing the *majority predicate* $\phi : \mathbb{R}^2 \to \{0, 1\}$ defined by $\phi(a, b) = 1$ if and only if $a > b$ is robustly decidable. We remark that such a predicate is in general not a detection predicate, showing that the class of robustly decidable predicates is strictly larger than the class of stably decidable one.

We first give an example of a CRD that "almost" decides the majority Let $\hat{\mathcal{D}} = (\hat{\Lambda}, \hat{R}, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \hat{\mathbf{i}})$ with $\Upsilon_{\text{yes}} = \{Y\}$ and $\Upsilon_{\text{no}} = \{N\}$, with initial state $\{aA, bB, 1Y\}$ and $\mathbf{i} = \{1Y\}$, and reactions

$$A + N \xrightarrow{k_1} A + Y \tag{2}$$

$$B + Y \xrightarrow{k_2} B + N \tag{3}$$

$$A + B \xrightarrow{k_3} \emptyset \tag{4}$$

$\hat{\mathcal{D}}$ both robustly (even stably) decides majority when $a \neq b$. Consider the case when $a > b$. Intuitively, only reaction (4) decreases the concentrations of A and B, and thus the concentration of A remains positive (as $a - b > 0$) while the concentration of B converges to zero. As a result, (2) will convert all of species N to species Y. Similarly, when $a < b$ as only concentration of B remains positive (3) converts all of Y to N.

Unfortunately, this CRD is unable to robustly decide majority when $a = b$. In this case, reaction (4) decreases both A and B at the same rate, allowing rate constants to influence the output. The correct output for this case is "no", but the adversary can choose $k_1 = 100, k_2 = 5, k_3 = 1$ so that when A and B approach 0, the rate of reaction (2) will be greater than reaction (3). Thus the CRD will incorrectly converge to $\{1Y\}$ and output "yes". Note that there exist rate constants under which the correct answer is reached. For example, when $k_1 < k_2$, the CRN converges to the state $\{1N\}$.

Since Theorem B.2 implies no CRD can *stably* decide majority, any modification to $\hat{\mathcal{D}}$ so that it *robustly* decides majority must take advantage of mass-action rate laws.

This is why the addition of the auxiliary species C is necessary in Algorithm 3.1. As C converges to zero *asymptotically* slower than A, its presence can speed up the production of N. This use of different asymptotic decays is a powerful tool of mass-action, not available for stable computation, which explains that a CRD can robustly decide majority but cannot stably decide it.

Algorithm 1 The following CRD robustly decides the majority predicate $\phi(a, b) = 1$ if and only if $a > b$. CRD $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \mathbf{i})$ is defined as follows. Let $\Upsilon_{\text{yes}} = \{Y\}$ and $\Upsilon_{\text{no}} = \{N\}$ be the yes voter and no voter respectively. Let the initial state be $\{aA, bB, 1Y, 1C\}$ where $\{1Y, 1C\}$ is the initial context $\mathbf{i}$. We add the following reactions:

$$A + N \xrightarrow{k_1} A + Y \tag{5}$$
$$B + Y \xrightarrow{k_2} B + N \tag{6}$$
$$A + B \xrightarrow{k_3} \emptyset \tag{7}$$
$$C + Y \xrightarrow{k_4} C + N \tag{8}$$
$$3C \xrightarrow{k_5} \emptyset \tag{9}$$

▶ **Lemma 3.3.** *The CRD in Algorithm 3.1 robustly decides the majority predicate.*

Proof. Here we show that the CRD given in Algorithm 3.1 robustly decides the majority predicate $\phi(a, b) = 1$ if and only if $a > b$. We consider the following 3 cases separately.

***a* > *b*:** To argue that we converge to a correct vote, we must show that $\lim_{t\to\infty} Y(t) = 1$. This is equivalent to showing $\lim_{t\to\infty} N(t) = 0$, as $Y(t) + N(t) = 1$ for all t. We first observe that the only reactions that change the concentration of species A, B, and C strictly decrease the concentration of each species. Therefore, as t approaches infinity, the limits of their concentrations satisfy $\lim_{t\to\infty} A(t) = a - b$ and $\lim_{t\to\infty} B(t) = \lim_{t\to\infty} C(t) = 0$, as the initial concentration of A is strictly greater than the concentration of B. [11]
We observe the CRC induces this mass-action ODE for the concentration of N: $N'(t) = -k_1A(t)N(t) + k_2B(t)Y(t) + k_3C(t)Y(t)$. Using the fact that $Y(t) + N(t) = 1$, we may rewrite the ODE in terms of only $N(t)$ and species whose limit is known: $N'(t) + N(t)(k_1A(t) + k_2B(t) + k_3C(t)) = k_2B(t) + k_3C(t)$. We now want to apply Lemma 3.1, to connect the asymptotic behavior of A, B and C to that of N. To see that this lemma

[11] This can also be seen by application of Lemma 2.9. The sub-CRN with reactions $A + B \to \emptyset$ and $3C \to \emptyset$ is feedforward and stably computes $f(a, b, c) = a - b$ when $a > b$, converging to the static state $\{(a - b)A, 0B, 0C\}$. By Lemma 2.9, this restricted CRN robustly computes $a - b$.

is applicable, we observe that with the initial conditions $A(0) = a > 0$, the function $A(t)$ is always positive as it monotonically decreases and approaches a positive limit. This also implies the lower bound $A(t) \geq a - b$, which is positive in this case. Further, $B(t)$ and $C(t)$ are both non-negative, so we can apply Lemma 3.1 with $f(t) = N(t)$, $p(t) = k_1A(t) + k_2B(t) + k_3C(t)$ and $g(t) = k_2B(t) + k_3C(t)$ to obtain the bound

$$\begin{aligned} N(t) &\leq \frac{k_2B(t) + k_3C(t)}{k_1A(t) + k_2B(t) + k_3C(t)} + K \exp\left(-\int_0^t k_1A(s) + k_2B(s) + k_3C(s)ds\right) \\ &\leq \frac{k_2B(t) + k_3C(t)}{k_1A(t) + k_2B(t) + k_3C(t)} + K \exp\left(-(a-b)k_1t\right) \end{aligned}$$

Since B and C converge to 0 as t approaches infinity, this shows $\lim_{t\to\infty} N(t) = 0$.

***a* < *b*:** Symmetric to the previous case.

***a* = *b*:** We want to show that in this case CRN converges to the static state $\mathbf{c} = \{1N\}$. With the initial conditions $A(0) = B(0) = a$, it can be found that the functions $A(t)$ and $B(t)$ are equal from their mass action ODEs $A'(t) = B'(t) = -A(t)B(t)$. We can then derive that the closed form for the concentrations $A(t)$ and $B(t)$ is $\frac{a}{k_1t+1}$, where a is the initial concentration of A. Similarly, we can compute that C has closed form $C(t) = \frac{1}{\sqrt{2k_3t+1}}$. We consider the mass action ODE for $Y'(t)$

$$Y'(t) + \left(\frac{2a}{k_1t+1} + \frac{1}{\sqrt{2k_3t+1}}\right) Y(t) = \frac{a}{k_1t+1}$$

This is a first-order linear ordinary differential equation, so we can find an explicit solution for Y by way of integrating factor:

$$Y(t) = \frac{\int_0^t (as+1)^{a/k_1} e^{\sqrt{2k_3s+1}/k_3} ds + 1}{(at+1)^{2a/k_1} e^{\sqrt{2k_3t+1}/k_3}}.$$

Taking the limit as $t \to \infty$, observe that the numerator and the denominator both approach ∞ for any positive choice of k_1 and k_3. Thus,

$$\begin{aligned} \lim_{t\to\infty} Y(t) &= \lim_{t\to\infty} Y'(t) && \text{L'Hôpital's rule} \\ &= \lim_{t\to\infty} \frac{\frac{a}{k_1t+1}}{\frac{2a}{k_1t+1} + \frac{1}{\sqrt{2k_3t+1}}} \\ &= \lim_{t\to\infty} \frac{a}{k_1t+1} \times \frac{(k_1t+1)\sqrt{2k_3t+1}}{2a\sqrt{2k_3t+1} + k_1t + 1} = \lim_{t\to\infty} \frac{O\left(t^{3/2}\right)}{O\left(t^2\right)} = 0. \end{aligned}$$

Since $Y(t)$ approaches 0 as $t \to \infty$, $N(t)$ approaches 1 as desired. ◀

Next, we show that threshold predicates are robustly decidable by CRDs.

▶ **Lemma 3.4.** *Every threshold predicate is robustly decidable by a continuous CRC.*

Proof sketch. A full proof is given in Section A. It works by reducing the problem of deciding ϕ to that of deciding majority as in Lemma 3.3. For example, to decide whether $2x_1 - x_2/3 + \frac{5}{4}x_3 > 4$, we start with $4B$, and we convert positive terms to A and negative terms to B with the correct rational multipliers: reactions $X_1 \to 2A$ and $4X_3 \to 5A$ for the positive terms and $3X_2 \to B$ for the negative term. The reactions of Lemma 3.3 then properly decide whether $A > B$, which, since the above reactions converge to $A = 2x_1 + \frac{5}{4}x_3$ and $B = 4 + x_2/3$, is true if and only if the threshold predicate $2x_1 - x_2/3 + \frac{5}{4}x_3 > 4$ holds. ◀

We will now show that Boolean combinations of threshold predicates are also robustly decidable. To show this result, we first prove a lemma that lets us assume CRDs that robustly compute predicates are of a convenient form: without loss of generality, we can assume that a CRD has exactly one yes and no voter. Furthermore, both of their concentrations converge exactly to 1 and 0 as $t \to \infty$.

▶ **Lemma 3.5.** *Let $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \mathbf{i})$ robustly decide the predicate $\phi : \mathbb{R}^k_{\geq 0} \to \{0,1\}$. Then there is a CRD $\mathcal{D}' = (\Lambda', R', \Sigma', \{Y\}, \{N\}, \mathbf{i}')$ that robustly decides ϕ with exactly one yes voter species Y and one no voter species N. Furthermore, the concentration of these voters satisfy $Y(t) + N(t) = 1$ for all $t \in \mathbb{R}_{\geq 0}$.*

Proof sketch. A full proof is given in Section A. For each original yes voter V_{y}, we add a reaction $V_{\text{y}} + N \to V_{\text{y}} + Y$, and similarly $V_{\text{n}} + Y \to V_{\text{n}} + N$, to influence the new voters in the correct direction. ◀

This result shows that Definition 2.6 is equivalent to a model in which we require exactly one yes voter and no voter. Furthermore, this lemma allows us to insist that correct voting species do not just remain above zero, but in fact converge to a particular value.

▶ **Lemma 3.6.** *Let $\phi_1 : \mathbb{R}^k_{\geq 0} \to \{0,1\}$ and $\phi_2 : \mathbb{R}^k_{\geq 0} \to \{0,1\}$ be robustly decidable predicates. Then the following are also robustly decidable: $\overline{\phi_1}$, $\phi_1 \wedge \phi_2$, and $\phi_1 \vee \phi_2$.*

Proof sketch. A full proof is given in [18]. To decide $\overline{\phi_1}$, it is enough to swap the yes and no voters. To decide $\phi_1 \wedge \phi_2$ or $\phi_1 \vee \phi_2$ we use the following construction. Similarly to Lemma 3.5, we add new voter species that are "influenced" by the voters of the CRDs deciding ϕ_1 and ϕ_2. The new voters are $V_{\text{yy}}, V_{\text{ny}}, V_{\text{yn}}, V_{\text{nn}}$, where the first subscript represents a vote for ϕ_1 and the second subscript a vote for ϕ_2. Yes-voters Y for ϕ_1 influence the new voters via $Y + V_{\text{ny}} \to Y + V_{\text{yy}}$ and $Y + V_{\text{nn}} \to Y + V_{\text{yn}}$ and no-voters N influence via $N + V_{\text{yy}} \to N + V_{\text{ny}}$ and $N + V_{\text{yn}} \to N + V_{\text{nn}}$. Voters for ϕ_2 similarly influence the second subscript. To decide $\phi_1 \vee \phi_2$, let $V_{\text{yy}}, V_{\text{ny}}, V_{\text{yn}}$ be the yes voters. To decide $\phi_1 \wedge \phi_2$, let V_{yy} be the yes voter. ◀

The following is the first main result of this paper.

▶ **Theorem 3.7.** *Every multi-threshold predicate is robustly decidable by a continuous CRD.*

Proof. Immediate from induction on the number of threshold predicates defining the multi-threshold predicate; Lemma 3.4 is the base case and Lemma 3.6 is the inductive case. ◀

3.2 Piecewise affine functions are robustly computable

The next definition captures the idea that a species S converges because reactions stop changing S, as opposed to reactions producing S at the same rate other reactions consume S.

▶ **Definition 3.8.** *Consider CRN $\mathcal{C} = (\Lambda, R)$ with initial state $\mathbf{x}$. Recall $\rho_t(\alpha)$ is the rate of $\alpha \in R$ at time t. We say that from $\mathbf{x}$, a species $S \in \Lambda$* approaches static steady state*, if for every reaction $\alpha = (\mathbf{r}, \mathbf{p}, k) \in R$ such that $\mathbf{r}(S) \neq \mathbf{p}(S)$, $\lim_{t\to\infty} \rho_t(\alpha) = 0$.*

Note that if S approaches static steady state then $\lim_{t\to\infty} S(t)$ exists and is finite. In other words, although the CRN may not converge to a single state (perhaps some species oscillate or diverge to ∞), not only does the value of $S(t)$ converge, a stronger condition holds: the CRN converges to rate 0 of every reaction net producing or net consuming S. This contrasts *dynamic steady state*, e.g., for $S \underset{1}{\overset{1}{\rightleftharpoons}} A$, where S approaches a limit, but the reactions producing/consuming S have positive rate even at steady state (where $S = A$).

▶ **Lemma 3.9.** *Every rational floor-affine function $f(\mathbf{x}) = \max(0, h + \sum_{i=1}^{k} w_i \cdot \mathbf{x}(i))$ is stably and robustly computable by a continuous feedforward CRC with output species Y, such that Y approaches static steady state and Y's concentration never exceeds $h + \sum_{i,w_i>0} w_i \cdot \mathbf{x}(i)$.*

Proof. Since all feedforward CRCs that stably compute a function also robustly compute it (Lemma 2.9), it suffices to define a feedforward CRC stably computing f. Define the CRC $\mathcal{C} = (\Lambda, R, \Sigma, Y, \mathbf{i})$, where $\Sigma = \{X_1, \ldots, X_k\}$, $\Lambda = \Sigma \cup \{Y, Y^-\}$, and $\mathbf{i}$ and R are defined below. Let $f(\mathbf{x}) = \max(0, g(\mathbf{x}))$ be rational floor-affine, with $g : \mathbb{R}^k_{\geq 0} \to \mathbb{R}$ a rational affine function $g(\mathbf{x}) = h + \sum_{i=1}^{k} \frac{n_i}{d_i} \cdot \mathbf{x}(i)$, for $h \in \mathbb{R}$, $d_1, \ldots, d_k \in \mathbb{Z}$, and $n_1, \ldots, n_k \in \mathbb{Z}_{>0}$; note we have written each rational w_i as $\frac{n_i}{d_i}$. Start with initial context $\mathbf{i} = \{hY\}$ if $h > 0$ and $\mathbf{i} = \{|h|Y^-\}$ if $h \leq 0$. For every i such that $d_i > 0$, add reaction $n_i X_i \to d_i Y$ to R, and for every i such that $d_i < 0$, add reaction $n_i X_i \to d_i Y^-$ to R. Finally we add reaction $Y + Y^- \to \emptyset$. Except for the last reaction, all reactions are entirely independent – none share a reactant with another reaction. Since the last reaction does not produce any species, $\mathcal{C}$ is feedforward.

From any reachable state, execute reactions of the form $n_i X_i \to d_i Y$ and $n_i X_i \to d_i Y^-$ until all inputs are gone. By inspection of the reactions, this means $\sum_{i,n_i>0} \frac{n_i}{d_i} \mathbf{x}(i)$ amount of Y has been produced in total by such reactions, and $\sum_{i,n_i<0} \frac{n_i}{d_i} \mathbf{x}(i)$ amount of Y^- has been produced in total by such reactions. Then run reaction $Y + Y^- \to \emptyset$ until both reactants are gone. If $g(\mathbf{x}) > 0$ then Y^- will be limiting and this will result in $\sum_{i,n_i>0} \frac{n_i}{d_i} \mathbf{x}(i) - \sum_{i,n_i<0} \frac{n_i}{d_i} \mathbf{x}(i) = \sum_{i=1}^{k} \frac{n_i}{d_i} \mathbf{x}(i) = g(\mathbf{x}) = f(\mathbf{x})$, and if $g(\mathbf{x}) \leq 0$ then Y will go to 0, so $\mathcal{C}$ stably computes $f(\mathbf{x}) = \max(0, g(\mathbf{x}))$.

Since $\mathcal{C}$ is feedforward, its steady state is static [20, Lemma 4.8]. Finally, the fact that Y cannot exceed $h + \sum_{i,w_i>0} w_i \cdot \mathbf{x}(i)$ follows from the fact that we start with hY (if $h > 0$) and the only reactions producing Y are $n_i X_i \to d_i Y$ for i such that $\frac{n_i}{d_i} = w_i > 0$. ◀

The following is the second main result of this paper.

▶ **Theorem 3.10.** *Every threshold-piecewise rational floor-affine function is robustly computable by a continuous CRC.*

Proof. This construction is similar to Lemma 4.4 in [21], though that proof was different (and simpler), applying to stable computation in the discrete CRN model. For a threshold-piecewise rational floor-affine function $f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$, recall the floor-affine functions $f_1, \ldots, f_l : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ and the multi-threshold predicates $\phi_1, \ldots, \phi_\ell : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ from Definition 2.3. We equivalently think of each ϕ_j as a set $R_j = \phi_j^{-1}(1)$ that makes ϕ_j true, i.e., f partitions the $\mathbb{R}^k_{\geq 0}$ into ℓ disjoint regions $R_1, \ldots, R_\ell$.

Let $j \in \{1, \ldots, \ell\}$. By Lemma 3.6 there is a CRD $\mathcal{D}_j = (\Lambda_j^{\mathcal{D}}, R_j^{\mathcal{D}}, \Sigma_j^{\mathcal{D}}, \{T_j\}, \{F_j\}, \mathbf{i}_j^{\mathcal{D}})$, robustly deciding ϕ_j, where we may assume a single yes voter T_j and single no voter F_j for each by Lemma 3.5. By Lemma 3.9 there is a CRC $\mathcal{C}_j = (\Lambda_j^{\mathcal{C}}, R_j^{\mathcal{C}}, \Sigma_j^{\mathcal{C}}, Y_j, \mathbf{i}_j^{\mathcal{C}})$ robustly computing f_j. We construct the CRC $\mathcal{C} = (\Lambda, R, \Sigma, Y^P, \mathbf{i})$ to robustly compute f, where $\Sigma = \{X_1, \ldots, X_k\}$. We transform the input species from Σ to those of each $\Sigma_i^{\mathcal{D}}, \Sigma_i^{\mathcal{C}}$ by adding reactions of the form, for each $1 \leq i \leq k$, $X_i \to X_i^{1,\mathcal{D}} + X_i^{1,\mathcal{C}} + \cdots + X_i^{\ell,\mathcal{D}} + X_i^{\ell,\mathcal{C}}$ (similar to the construction in the proof of Lemma 3.6).

Ideally we would convert the correct "local" output Y_j to the "global" output Y, but $\mathcal{C}_j$ may need to consume Y_j; in fact it provably must do so if computing a non-monotone function such as $x_1 - x_2$. To avoid interfering with $\mathcal{C}_j$'s computation, we instead modify its reactions to have additional products that will be used in the conversion. Since we do not change any reactants, and since the new products are new species not appearing in $\mathcal{C}_j$, these new products will preserve the kinetics of the CRC $\mathcal{C}_j$, while enabling the constructed CRC to "read" the output of $\mathcal{C}_j$.[12]

[12] We use a trick known as *dual-rail encoding* [21, 22]; we use it more like [21] as a handy "intermediate" proof technique, not as in [20] in which inputs and outputs of CRCs are encoded in dual-rail.

For each reaction net producing p copies of Y_j in $\mathcal{C}_j$, for example $A + Y_j \to B + 3Y_j$ that net produces 2 Y_j, we modify the reaction to also produce p copies of a new species Y_j^P: $A + Y_j \to B + 3Y_j + 2Y_j^P$. For each reaction net consuming c copies of Y_j in $\mathcal{C}_j$, for example $B + 4Y_j \to Y_j$ that net consumes 3 Y_j, we modify the reaction to also produce c copies of a new species Y_j^C: $B + 4Y_j \to Y_j + 3Y_j^C$. This maintains that $Y_j(t) = Y_j^P(t) - Y_j^C(t)$ for all t. Y_j^P "records" the total amount of Y_j that has ever been produced by any reaction, and Y_j^C the total amount of Y_j that has ever been consumed, so that their difference is the net amount of Y_j produced. If $\mathbf{i}_j^{\mathcal{C}}(Y_j) > 0$, we also modify $\mathbf{i}$ so that $\mathbf{i}(Y_j^P) = \mathbf{i}_j(Y_j)$, i.e., if $\mathcal{C}_j$ starts with some Y_j already "produced", we want to start with the same amount of Y_j^P to maintain $Y_j(t) = Y_j^P(t) - Y_j^C(t)$. (captured by the multiset term $\mathbf{y}$ below)

Let $\Lambda = \{Y^P, Y^C\} \cup \Sigma \cup \bigcup_{j=1}^{\ell} \Lambda_j^{\mathcal{D}} \cup \Lambda_j^{\mathcal{C}} \cup \{Y_j^P, Y_j^C\}$, and $\mathbf{i} = \sum_{j=1}^{\ell} \mathbf{i}_j^{\mathcal{D}} + \mathbf{i}_j^{\mathcal{C}} + \mathbf{y}$, where $\mathbf{y}(Y_j^P) = \mathbf{i}_j^{\mathcal{C}}(Y_j)$ and $\mathbf{y}(S) = 0$ for all other $S \in \Lambda$.

R includes all reactions from each $\mathcal{C}_j$ and $\mathcal{D}_j$, after modifying some as explained above, and the following reactions for each $1 \leq j \leq \ell$:

$$T_j + Y_j^P \xrightarrow{k_{j_1}} T_j + \hat{Y}_j^P + Y^P \tag{10}$$

$$T_j + Y_j^C \xrightarrow{k_{j_2}} T_j + \hat{Y}_j^C + Y^C \tag{11}$$

$$F_j + \hat{Y}_j^P \xrightarrow{k_{j_3}} F_j^P + Y_j^P + Y^C \tag{12}$$

$$F_j + \hat{Y}_j^C \xrightarrow{k_{j_4}} F_j^C + Y_j^C + Y^P \tag{13}$$

$$Y^P + Y^C \xrightarrow{k} \emptyset \tag{14}$$

We think of the outputs Y_i^P, Y_i^C produced by $\mathcal{C}_i$ as "inactive", with "active" versions $\hat{Y}_i^P, \hat{Y}_i^C$. Then T_j activates them and F_j deactivates them, while maintaining that as much Y^P, Y^C is produced as the amount of $\hat{Y}_j^P, \hat{Y}_j^C$ that has been net activated by the voters T_j, F_j. Furthermore, to ease analysis of the ODEs, we ensure that Y^P and Y^C do not influence the reaction rates. For, say, reaction (12) to reverse the effect of reaction (10) straightforwardly, it would consume Y^P as a reactant. However, with dual rail it is equivalent to produce Y^C, simplifying the expression for the rate of (12).

By inspection of reactions (10), (11), (12), (13), (14), as well as the modified reactions of $\mathcal{C}_j$, one can verify the conservation law that for all t,

$$\sum_{j=1}^{\ell} \hat{Y}_j^P(t) - \sum_{j=1}^{\ell} \hat{Y}_j^C(t) = Y^P(t) - Y^C(t). \tag{15}$$

since each reaction changes each side of (15) by the same amount. Similarly, the following conservation law holds for each $1 \leq j \leq \ell$:

$$Y_j^P(t) - Y_j^C(t) + \hat{Y}_j^P(t) - \hat{Y}_j^C(t) = Y_j(t). \tag{16}$$

This completes the description of the construction of $\mathcal{C}$.

We now prove that $\mathcal{C}$ robustly computes f. Let $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$. Assume without loss of generality that $\phi_1(\mathbf{x}) = 1$ (thus $\phi_j(\mathbf{x}) = 0$ for all $j > 1$), i.e., that $f(\mathbf{x}) = f_1(\mathbf{x})$, so that our goal is to converge to activating all $\hat{Y}_1^P, \hat{Y}_1^C$ and deactivating all $\hat{Y}_j^P, \hat{Y}_j^C$ for $j > 1$. Since each $\mathcal{D}_j$ robustly decides ϕ_j, $\lim_{t\to\infty} T_1 = 1$, $\lim_{t\to\infty} F_1 = 0$, and for all $j > 1$, $\lim_{t\to\infty} T_j = 0$, $\lim_{t\to\infty} F_j = 1$.

By the conservation law shown above (15), to demonstrate that $\lim_{t\to\infty} Y^P(t) = f_1(\mathbf{x})$, it suffices to show that $\hat{Y}_1^P(t) - \hat{Y}_1^C(t)$ approaches $f_1(\mathbf{x})$ while $\hat{Y}_j^P(t)$ and $\hat{Y}_j^C(t)$ approach 0 for all $j > 1$. We first will show that for all $j > 1$, $\hat{Y}_j^P(t)$ and $\hat{Y}_j^C(t)$ approach 0. For each j,

the ODE for $\hat{Y}_j^P$ is given by $\frac{d}{dt}\hat{Y}_1^P = k_{j_1}T_jY_j^P - k_{j_3}F_j\hat{Y}_j^P$. We apply Lemma 3.2 to obtain that $\hat{Y}_j^P$ is bounded as

$$\hat{Y}_j^P \leq \frac{5k_{j_1}T_j(t)Y_j^P(t)}{k_{j_3}F_j(t)+(t^2+1)^{-1}} + Kk_{j_3}\exp\left(-\int_0^t F_j(s)ds\right).$$

By the correctness of $\mathcal{D}_j$, for all $j > 1$ the concentrations $T_j(t)$ and $F_j(t)$ approach 0 and 1 respectively as t approaches infinity. Further, the function $Y_j^P(t)$ is bounded above by the construction in Lemma 3.9. Hence, the quotient term converges to zero as t approaches infinity. We also observe that $F_j(t)$ approaching 1 implies that the integral $\int_0^t F_j(s)ds$ diverges as $t \to \infty$, which shows that the exponential term also converges to 0. This demonstrates that $\lim_{t\to\infty}\hat{Y}_j^P(t) = 0$, and a similar argument with the ODE for $\hat{Y}_j^C(t)$ shows that $\lim_{t\to\infty}\hat{Y}_j^C(t) = 0$ as well. To show that $\hat{Y}_1^P(t) - \hat{Y}_1^C(t)$ approaches $f_1(\mathbf{x})$, we appeal to the second conservation law derived (16). Since $\mathcal{C}_1$ robustly computes f_1, the limit $\lim_{t\to\infty}Y_1(t) = f_1(\mathbf{x})$. Hence, to show the result we can show simply show that $Y_j^P(t)$ and $Y_j^C(t)$ approach zero. Indeed, the ODE for $Y_j^P(t)$ is given by $\frac{d}{dt}Y_1^P(t) = k_3F_1(t)\hat{Y}_1^P(t) + P_1(\Lambda_1,t) - k_1T_1(t)Y_1^P(t)$ where $P_1(\Lambda_1,t)$ is a function determined by the production of Y_1^P in the CRC $\mathcal{C}_1$. We apply Lemma 3.2 to the ODE with $p(t) = T_1(t)$ and $g(t) = F_1(t)\hat{Y}_1^P(t) + P(\Lambda_1,t)$ to obtain the bound $Y_1^P(t) \leq \frac{k_3F_1(t)\hat{Y}_1^P(t)+P_1(\Lambda_1,t)}{k_1T_1(t)+(t^2+1)^{-1}} + K\exp\left(-\int_0^t k_1T_1(s)ds\right)$. The construction given in Lemma 3.9 has Y_1 approach a static steady state, so the rate of production $P_1(\Lambda_1,t)$ tends to zero as $t \to \infty$. Furthermore, and duel to the previous case, the correctness of $\mathcal{D}_1$ shows that $F_1(t) \to 0$ and $T_1(t) \to 1$, so $Y_1^P(t) \to 0$. Similar arguments also demonstrate that $Y_1^C(t) \to 0$ as well. To complete the result, we note that by inspection of the reactions, at any time t we have $Y^P(t) \geq Y^C(t)$, so reaction (14) causes $Y^C(t) \to 0$. Putting it all together, we have shown the desired limit:

$$\begin{aligned}
\lim_{t\to\infty} Y^P(t) &= \lim_{t\to\infty}\left(\sum_{j=1}^{\ell}\hat{Y}_j^P(t) - \sum_{j=1}^{\ell}\hat{Y}_j^C(t) + Y^C(t)\right) \\
&= \lim_{t\to\infty}\left(\hat{Y}_1^P(t) - \hat{Y}_1^C(t)\right) + \lim_{t\to\infty}\left(\sum_{j=2}^{\ell}\hat{Y}_j^P(t) - \sum_{j=2}^{\ell}\hat{Y}_j^C(t)\right) + \lim_{t\to\infty}Y^C(t) \\
&= f_1(\mathbf{x}) + 0 + 0 = f(\mathbf{x}).
\end{aligned}$$

◀

4 Conclusion

Motivated by the limitations of stable predicate computation, we investigated the robust computation of predicates and numerical functions. While we established positive results on what can be robustly computed, namely multi-threshold predicates (Definition 2.2) and robustly piecewise floor-affine functions (Definition 2.3), the limitations of robust computation remains an open question. We conjecture the positive results are tight, i.e., *exactly* the multi-threshold predicates and robustly piecewise floor-affine functions are robustly computable.

We have assumed the presence of initial context, for example to help include a positive amount of some voter species in all of our constructions, which gave the nice invariant that the sum of voter concentrations is preserved. This choice seems as though it is not strictly necessary, since our construction could instead generate voter species. We conjecture that a leaderless model, without initial context, would only restrict threshold predicates to a constant threshold of 0 and constrain functions to piecewise rational floor-*linear* functions, rather than affine.

Another possible extension of this work is employing the so-called *dual-rail* representation [20] to accommodate negative values as inputs. This means that a (possibly negative) value x is represented as the difference between two nonnegative species concentrations $X^+ - X^-$. This is known to slightly reduce the class of stably computable functions $f : \mathbb{R}^k \to \mathbb{R}$; such a function is stably computable using the dual-rail representation (for both inputs and output) if and only if it is piecewise rational linear and *continuous*.

In the construction given in Theorem 3.10, we employed a technique to modify upstream CRCs, ensuring controlled composition even when certain reactions within these CRCs consumed the output species. This structured approach to dependency management could naturally extend to the broader tricky problem of CRN composition [19, 30, 34].

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L Rivest. Time-space trade-offs in population protocols. In *Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms*, pages 2560–2579. SIAM, 2017. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-optimal majority in population protocols. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2221–2239. SIAM, 2018. doi:10.1137/1.9781611975031.144.

3 Dan Alistarh, Rati Gelashvili, and Milan Vojnović. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 47–56, 2015. doi:10.1145/2767386.2767429.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

5 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006. doi:10.1007/S00446-005-0138-3.

6 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, 2006. doi:10.1145/1146381.1146425.

7 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21:183–199, 2008. doi:10.1007/S00446-008-0067-Z.

8 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008. doi:10.1007/S00446-008-0059-Z.

9 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. doi:10.1007/S00446-007-0040-2.

10 Guillaume Aupy and Olivier Bournez. On the number of binary-minded individuals required to compute sqrt(1/2). *Theoretical Computer Science*, 412(22):2262–2267, 2011. doi:10.1016/j.tcs.2011.01.003.

11 Stav Ben-Nun, Tsvi Kopelowitz, Matan Kraus, and Ely Porat. An o (log3/2 n) parallel time population protocol for majority with o (log n) states. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 191–199, 2020. doi:10.1145/3382734.3405747.

12 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with o (log5/3 n) stabilization time and θ (log n) states. In *32nd International Symposium on Distributed Computing, DISC 2018*, page 10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2018. doi:10.4230/LIPIcs.DISC.2018.10.

13 Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. Time-space trade-offs in population protocols for the majority problem. *Distributed Computing*, 34(2):91–111, 2021. doi:10.1007/S00446-020-00385-0.

14 Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with o (log2 n) states and o (log2 n) convergence time. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 451–453, 2017. doi:10.1145/3087801.3087858.

15 Olivier Bournez, Philippe Chassaing, Johanne Cohen, Lucas Gerin, and Xavier Koegler. On the convergence of population protocols when population goes to infinity. *Applied Mathematics and Computation*, 215(4):1340–1350, 2009. doi:10.1016/j.amc.2009.04.056.

16 Olivier Bournez, Pierre Fraigniaud, and Xavier Koegler. Computing with large populations using interactions. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464, pages 234–246, 2012. doi:10.1007/978-3-642-32589-2_23.

17 Olivier Bournez, Daniel S Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length. *Journal of the ACM (JACM)*, 64(6):1–76, 2017. doi:10.1145/3127496.

18 Kim Calabrese, David Doty, and Mina Latifi. Robust predicate and function computation in continuous chemical reaction networks, 2025. doi:10.48550/arXiv.2506.06590.

19 Cameron Chalk, Niels Kornerup, Wyatt Reeves, and David Soloveichik. Composable rate-independent computation in continuous chemical reaction networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(1):250–260, 2019.

20 Ho-Lin Chen, David Doty, Wyatt Reeves, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. *Journal of the ACM*, 70(3), May 2023. doi:10.1145/3590776.

21 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2014. Special issue of invited papers from DNA 2012. doi:10.1007/s11047-013-9393-6.

22 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature nanotechnology*, 8(10):755–762, 2013.

23 Rachel Cummings, David Doty, and David Soloveichik. Probability 1 computation with chemical reaction networks. *Natural Computing*, 15(2):245–261, 2016. Special issue of invited papers from DNA 2014. doi:10.1007/s11047-015-9501-x.

24 David Doty, Mahsa Eftekhari, Leszek Gąsieniec, Eric Severson, Przemyslaw Uznański, and Grzegorz Stachowiak. A time and space optimal stable population protocol solving exact majority. In *2021 IEEE 62nd annual symposium on foundations of computer science (FOCS)*, pages 1044–1055. IEEE, 2022.

25 David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. *Natural Computing*, 14(2):213–223, 2015. Preliminary version appeared in DNA 2013. doi:10.1007/S11047-014-9435-8.

26 Moez Draief and Milan Vojnović. Convergence speed of binary interval consensus. *SIAM Journal on control and Optimization*, 50(3):1087–1109, 2012. doi:10.1137/110823018.

27 François Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *Computational Methods in Systems Biology: 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27–29, 2017, Proceedings 15*, pages 108–127. Springer, 2017. doi:10.1007/978-3-319-67471-1_7.

28 Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

29 Cato M. Guldberg and Peter Waage. Studies concerning affinity. *Forhandlinger: Videnskabs-Selskabet i Christinia. Norwegian Academy of Science and Letters*, 35, 1864. English translation in [39].

30 Hooman Hashemi, Ben Chugg, and Anne Condon. Composable Computation in Leaderless, Discrete Chemical Reaction Networks. In *DNA 26: 26th International Conference on DNA Computing and Molecular Programming*, volume 174, pages 3:1–3:18, 2020. doi:10.4230/LIPIcs.DNA.2020.3.

31 George B Mertzios, Sotiris E Nikoletseas, Christoforos L Raptopoulos, and Paul G Spirakis. Determining majority in networks with local interactions and very small local memory. *Distributed Computing*, 30(1):1–16, 2017. doi:10.1007/S00446-016-0277-8.

32 Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 35–42. IEEE, 2015. doi:10.1109/NCA.2015.35.

33 Yves Mocquard, Emmanuelle Anceaume, and Bruno Sericola. Optimal proportion computation with population protocols. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 216–223. IEEE, 2016. doi:10.1109/NCA.2016.7778621.

34 Eric E Severson, David Haley, and David Doty. Composable computation in discrete chemical reaction networks. *Distributed Computing*, 34(6):437–461, 2021. special issue of invited papers from PODC 2019. doi:10.1007/S00446-020-00378-Z.

35 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7:615–633, 2008. doi:10.1007/S11047-008-9067-Y.

36 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

37 PG Spirakis, L Gasieniec, Russell Martin, D Hamilton, and G Stachowiac. Deterministic population protocols for exact majority and plurality. *LIPIcs*, 70:14–1, 2017. doi:10.4230/LIPIcs.OPODIS.2016.14.

38 Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358(6369):eaal2052, 2017.

39 Peter Waage and Cato Maximilian Gulberg. Studies concerning affinity. *Journal of chemical education*, 63(12):1044, 1986. English translation; original paper is [29].

A Proofs for Section 3 (Robust computation by continuous CRNs)

▶ **Lemma 3.4.** *Every threshold predicate is robustly decidable by a continuous CRC.*

Proof. Intuitively, this construction translates the threshold problem to a majority problem. We translate the positive and negative contributions in the weighted sum to concentrations of species A and B respectively. We then use the reactions from the majority problem to detect if $A > B$ (with the threshold h added appropriately to either A or B).

Let $\phi : \mathbb{R}^k_{\geq 0} \to \{0,1\}$ be a threshold predicate. Without loss of generality that we assume all weights are integers by clearing denominators, so $\phi(\mathbf{x}) = 1 \iff \sum_{i=1}^{k} w_i \mathbf{x}(i) > h$ with each $w_i \in \mathbb{Z}$ and $h \in \mathbb{R}$. We construct the CRD $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}} \Upsilon_{\text{no}}, \mathbf{i})$ as follows. $\Sigma = \{X_1, \ldots, X_k\}$, and Λ is implicitly all species described below. Let $\Upsilon_{\text{yes}} = \{Y\}$ and $\Upsilon_{\text{no}} = \{N\}$ be the yes and no voter species respectively. Let the $\mathbf{i} = \{1Y, 1C, |h|AB\}$, where $AB = A$ if $h > 0$ and $AB = B$ otherwise. We then add the following reactions. For any $w_p > 0$ add reaction $X_p \to w_p A$ and for any $w_n < 0$ add reaction $X_n \to |w_n| B$. We also add the reactions from the CRD computing majority in Lemma 3.3.

Without loss of generality, we may insist that the threshold value h is zero as the reactions $L \to hB$ and $L \to |h|A$ influence the concentration of A and B respectively in the "same way" as the reactions $X_i \to w_i(A/B)$ in the sense that the concentration of each X_i species

and L have closed form solutions of decaying exponential functions. Let $\mathcal{P}$ be the set of all p for which $w_p > 0$ and let $\mathcal{N}$ be the set of all n for which $w_n < 0$. Let $a = \sum_{p\in\mathcal{P}} w_p \cdot x_p$ and $b = \sum_{n\in\mathcal{N}} |w_n| \cdot x_n$. We consider three cases.

$a > b$: It suffices to observe that $\lim_{t\to\infty} A(t) = a - b$ and $\lim_{t\to\infty} B(t) = 0$ since the restricted CRN that only contains species that change the concentration of A and B is feed-forward. A similar argument to the one used to prove the $a > b$ case of Lemma 3.3 then shows the result.

$a < b$: Symmetric to the previous case.

$a = b$: Our goal is to show that the concentration of species N converges to 1. Equivalently, we will show that the concentration of the yes voter Y converges to zero as the invariant $Y(t)+N(t) = 1$ is held for all t. The following argument will be very similar to the one we used to prove the correctness of our majority CRD. The CRD induces the following ODE for the concentration $Y(t)$: $Y'(t) = k_1A(t)N(t) - k_2B(t)Y(t) - k_4C(t)Y(t)$. Using the fact that $Y(t)+N(t) = 1$ we can rewrite this ODE as $Y'(t)+Y(t)[k_1A(t)+k_2B(t)+k_4C(t)] = k_1A(t)$. From the initial value problem $C'(t) = -C(t)^3$ with $C(0) = 1$, we obtain that a closed form for the concentration of C is $C(t) = (2k_4t + 1)^{-1/2}$. Hence $C(t)$ is strictly positive, so we can apply Lemma 3.1 to find that $Y(t) \leq \frac{k_1A(t)}{k_1A(t)+k_2B(t)+k_4C(t)} + K\exp\left(-\int_0^t k_1A(s) + k_2B(s) + k_4C(s)ds\right)$. With the observation that $B(t) \geq 0$ and $A(t) \geq 0$ for all $t \in \mathbb{R}_{\geq 0}$, and substituting in the closed form for $C(t)$, we obtain the bound $Y(t) \leq \frac{k_1A(t)}{k_4C(t)} + K\exp\left(-(2k_4^2t + k_4)\right))$. The exponential term clearly converges to zero, so to prove the result it suffices to show the quotient term converges to zero. We observe that the restricted CRN with only reactions that change the concentration of A and B is the feed-forward CRN:

$$\begin{aligned} \forall p \in \mathcal{P} : X_p &\to w_pA \\ \forall n \in \mathcal{N} : X_n &\to |w_n|B \\ A + B &\to \emptyset \end{aligned}$$

This CRN stably computes the function $f(a, b) = \min(a-b, 0)$, which by [20] Corollary 4.11 implies that it robustly computes $\min(a - b, 0)$. This shows that $\lim_{t\to\infty} A(t) = a - b = 0$. Taking the limit of the quotient, we obtain $0/0$, a so-called indeterminate form. Hence, we may apply L'Hôpital's rule to obtain the following estimate.

$$\begin{aligned} \lim_{t\to\infty} \frac{k_1}{k_4}\frac{A(t)}{C(t)} &= \frac{k_1}{k_4}\lim_{t\to\infty}\frac{A(t)'}{C(t)'} = \frac{k_1}{k_4}\lim_{t\to\infty}\frac{\sum_{p\in\mathcal{P}} k_pX_p - k_3A(t)B(t)}{(2k_4t+1)^{-3/2}} \\ &\leq \frac{k_1}{k_4}\lim_{t\to\infty}\frac{\sum_{p\in\mathcal{P}} k_pX_p}{(2k_4t+1)^{-3/2}} \end{aligned}$$

The equation for $A'(t)$ is from its mass action differential equation. Each input species X_i can be found to have a closed form of $w_ix_ik_ie^{-k_pt}$, which shows that the quotient tends to zero as t goes to infinity. Thus, $\lim_{t\to\infty} Y(t) = 0$ as desired. ◀

▶ **Lemma 3.5.** *Let $\mathcal{D} = (\Lambda, R, \Sigma, \Upsilon_{\text{yes}}, \Upsilon_{\text{no}}, \mathbf{i})$ robustly decide the predicate $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$. Then there is a CRD $\mathcal{D}' = (\Lambda', R', \Sigma', \{Y\}, \{N\}, \mathbf{i}')$ that robustly decides ϕ with exactly one yes voter species Y and one no voter species N. Furthermore, the concentration of these voters satisfy $Y(t) + N(t) = 1$ for all $t \in \mathbb{R}_{\geq 0}$.*

Proof. We construct the CRD $\mathcal{D}'$ as follows: Let $\Sigma' = \Sigma$, $\Lambda' = \Lambda \cup \{Y, N\}$. Keep all reactions from the original CRD, and for each yes voter $V_{\text{y}} \in \Upsilon_{\text{yes}}$, add the reaction $V_{\text{y}} + N \xrightarrow{k_{V_{\text{y}}}} V_{\text{y}} + Y$.

Additionally, for each no voter $V_\mathrm{n} \in \Upsilon_\mathrm{no}$ add the reaction $V_\mathrm{n} + Y \overset{k_{V_\mathrm{n}}}{\to} V_\mathrm{n} + N$. Start with initial context $\mathbf{i}' = \mathbf{i} + \{1Y\}$.

To show that $\mathcal{D}'$ robustly decides ϕ, we will show that for all $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$ such that $\phi(\mathbf{x}) = 1$, the concentration of species Y approaches 1 as t approaches infinity when the initial state is $\mathbf{x} + \mathbf{i} + \{1Y\}$. The case when $\phi(\mathbf{x}) = 0$ is a similar argument.

We make the worst-case assumption that the adversary chooses $k_\mathrm{y} = \min_{V_\mathrm{y} \in \Upsilon_\mathrm{yes}} k_{V_\mathrm{y}}$ to be the same (and smallest) rate constant for all reactions of the form $V_\mathrm{y} + N \overset{k_{V_\mathrm{y}}}{\to} V_\mathrm{y} + Y$, and similarly chooses the rate constant $k_\mathrm{n} = \max_{V_\mathrm{n} \in \Upsilon_\mathrm{no}} k_{V_\mathrm{n}}$ for all reactions of the form $V_\mathrm{n} + Y \overset{k_{V_\mathrm{n}}}{\to} V_\mathrm{n} + N$. This is a safe worst-case assumption, since choosing larger rate constants for reaction $V_\mathrm{y} + N \overset{k_{V_\mathrm{y}}}{\to} V_\mathrm{y} + Y$, or smaller rate constants for $V_\mathrm{n} + Y \overset{k_{V_\mathrm{n}}}{\to} V_\mathrm{n} + N$ would only make the network converge faster to the correct values. Furthermore, scaling all rate constants in the system by the same factor modifies only the timescale of the mass-action trajectory of the ODEs, while preserving its shape and eventual convergence. To simplify the system, we scale all rate constants by $1/k_\mathrm{y}$, ensuring that $k_\mathrm{y} = 1$.

The CRD $\mathcal{D}'$ induces the following ODE for the concentration of the species N: $N'(t) + N(t)(k_\mathrm{n} V_\mathrm{n}(t) + k_\mathrm{y} V_\mathrm{y}(t)) = k_\mathrm{n} V_\mathrm{n}(t)$ We may then apply Lemma 3.2 to obtain the bound $N(t) \leq \frac{5 k_\mathrm{n} V_\mathrm{n}(t)}{k_\mathrm{n} V_\mathrm{n}(t) + k_\mathrm{y} V_\mathrm{y}(t) + (t^2+1)^{-1}} + K \exp\left(-\int_0^t k_\mathrm{n} V_\mathrm{n}(s) + k_\mathrm{y} V_\mathrm{y}(s) ds\right)$. By the correctness of $\mathcal{C}$, the concentration $V_\mathrm{y}(t)$ remains strictly positive as $t \to \infty$. Hence, the integral $\int_0^t k_\mathrm{n} V_\mathrm{n}(s) + k_\mathrm{y} V_\mathrm{y}(s)\, ds$ diverges to infinity as $t \to \infty$. This demonstrates that the exponential term tends to zero as t grows large. To see the quotient term goes to zero, the correctness of $\mathcal{C}$ also dictates that $V_\mathrm{n}(t) \to 0$ as $t \to \infty$. This shows that $N(t) \to 0$. ◀

B Stable predicate computation by continuous CRNs

In this section, we show that the class of predicates stably decidable by continuous CRNs is exactly the detection predicates. Intuitively, $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ is a *detection predicate* if it is a Boolean combination of questions in the form "is initial concentration of species S positive or not?". We formalize this notion as follows.

▶ **Definition B.1.** *A predicate $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ is a* simple detection predicate *if there is a $1 \leq i \leq k$ such that the extension of ϕ (the set $\phi^{-1}(1)$ of all input vectors that make ϕ true) is of the form $\{\mathbf{x} \in \mathbb{R}^k_{\geq 0} \mid \mathbf{x}(i) > 0\}$. A* detection predicate *is one expressible as a finite combination of ANDs, ORs, and NOTs of simple detection predicates.*

In other words, a simple detection predicate is defined by a hyperplane (with rational slopes), with output 1 on one side of the hyperplane and output 0 on the other side and on the hyperplane itself. A detection predicate ϕ is defined by a finite number of hyperplanes that partition $\mathbb{R}^k_{\geq 0}$ into a finite number of regions, and ϕ is constant within each region.

The following is main result of this section. Each direction is proven separately in Sections B.2 and B.3 via Lemmas B.8 and B.11.

▶ **Theorem B.2.** *$\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ is stably decidable by a continuous CRN if and only if ϕ is a detection predicate.*

B.1 Stable (rate-independent) computation

These definitions are taken from [20]; see Section 2.4 of that paper for justification that the notion of segment-reachability in particular (Definition B.4) is reasonable.

▶ **Definition B.3.** *state* $\mathbf{d}$ *is* straight-line reachable (aka 1-segment reachable) *from state* $\mathbf{c}$*, written* $\mathbf{c} \to^1 \mathbf{d}$*, if* $(\exists \mathbf{u} \in \mathbb{R}^R_{\geq 0})$ $\mathbf{c} + \mathbf{M}\mathbf{u} = \mathbf{d}$ *and* $\mathbf{u}(\alpha) > 0$ *only if reaction* α *is applicable at* $\mathbf{c}$*. In this case write* $\mathbf{c} \to^1_{\mathbf{u}} \mathbf{d}$*.*

Intuitively, by a single segment we mean running the reactions applicable at $\mathbf{c}$ at a constant (possibly 0) rate to get from $\mathbf{c}$ to $\mathbf{d}$. In the definition, $\mathbf{u}(\alpha)$ represents the flux, or total amount executed, of reaction $\alpha \in R$.

▶ **Definition B.4.** *Let* $k \in \mathbb{N}$*. state* $\mathbf{d}$ *is* k-segment reachable *from state* $\mathbf{c}$*, written* $\mathbf{c} \rightsquigarrow^k \mathbf{d}$*, if* $(\exists \mathbf{b}_0, \ldots, \mathbf{b}_k)$ $\mathbf{c} = \mathbf{b}_0 \to^1 \mathbf{b}_1 \to^1 \mathbf{b}_2 \to^1 \ldots \to^1 \mathbf{b}_k$*, with* $\mathbf{b}_k = \mathbf{d}$*. State* $\mathbf{d}$ *is* segment-reachable *(or simply* reachable*) from state* $\mathbf{c}$*, written* $\mathbf{c} \rightsquigarrow \mathbf{d}$*, if* $(\exists k \in \mathbb{N})$ $\mathbf{c} \rightsquigarrow^k \mathbf{d}$*.*

Often Definition B.4 is used implicitly, when we make statements such as, "Run reaction 1 until X is gone, then run reaction 2 until Y is gone", which implicitly defines two straight lines in concentration space.

We now formalize what it means for a CRN to "rate-independently" compute (stably decide) a predicate ϕ. We define the *global output partial function* $\Phi : \mathbb{N}^\Lambda \dashrightarrow \{0, 1\}$ as $\Phi(\mathbf{x}) = 1$. If the only voter species with positive concentration in the state $\mathbf{x}$ are yes voters, then $\Phi(\mathbf{x}) = 1$. Conversely, if the only voter species with positive concentration are no voters, $\Phi(\mathbf{x}) = 0$. Lastly, if neither of these conditions is met, the output function $\Phi(\mathbf{x})$ is undefined. We say a state $\mathbf{o}$ is *stable* if, for all $\mathbf{c}$ such that $\mathbf{o} \rightsquigarrow \mathbf{c}$: $\Phi(\mathbf{o}) = \Phi(\mathbf{c})$.

▶ **Definition B.5** (stably decide)**.** *Let* $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$ *be a predicate. We say a CRD* $\mathcal{D}$ stably decides ϕ *if, for all* $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$*, and all* $\mathbf{c}$ *such that* $\mathbf{x} \rightsquigarrow \mathbf{c}$*, there exists a stable state* $\mathbf{o}$ *such that* $\mathbf{c} \rightsquigarrow \mathbf{o}$ *and* $\Phi(\mathbf{o}) = \phi(\mathbf{x})$*. We say a set* A *is* stably decidable *if its indicator function* χ_A *is stably decidable.*

▶ **Definition B.6** (stably compute)**.** *Let* $f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}$ *be a function. We say a CRC* $\mathcal{C} = (\Lambda, R, \Sigma, \{Y\}, \sigma)$ stably computes f *if for all* $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$ *and all* $\mathbf{c}$ *such that* $\mathbf{x} \rightsquigarrow \mathbf{c}$*, there exists a stable state* $\mathbf{o}$ *such that* $\mathbf{c} \rightsquigarrow \mathbf{o}$ *and* $\mathbf{o}(Y) = f(\mathbf{x})$*.*

▶ **Theorem B.7.** *A function* $f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}$ *is stably computable by a continuous CRC if and only if it is positive-continuous and piecewise rational linear.*

A linear function's graph defines a k-dimensional hyperplane in $\mathbb{R}^{k+1}_{\geq 0}$ that passes through the origin (i.e., linear but not affine). If a piecewise rational linear function is positive-continuous, we switch from one linear component f_i to another f_j only where their hyperplanes intersect. Thus the question "*is linear component* f_i *the correct linear component to apply to compute* f *on input* $\mathbf{x}$*?*" is itself a *multi-threshold predicates* as defined in Definition 2.2, but with constant $h = 0$.

B.2 Positive result: All detection predicates are stably decidable by continuous CRNs

We first show that continuous CRNs can stability decide all detection predicates.

▶ **Lemma B.8.** *Every detection predicate is stably decidable by a continuous CRN.*

Proof. To decide the detection predicate $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$, we let $\Sigma = \{X_{\{1\}}, X_{\{2\}}, \ldots, X_{\{k\}}\}$ and let $\Lambda = \{X_U \mid U \subseteq \{1, \ldots, k\}, U \neq \emptyset\}$ be the set of species. The set of yes voters Υ_{yes} is the set of all X_U such that $\phi(\mathbf{x}) = 1$ for any $\mathbf{x}$ with $\text{supp}(\mathbf{x}) = U$ (note this is well-defined since ϕ is a detection predicate; it must have the same output on all $\mathbf{x}$ with identical support) and the set of no voters be each other species. The reactions in the CRN are

$X_U + X_T \to X_{U \cup T}$. This stably decides ϕ, because eventually all species become X_{U^*} where U^* represents the set of species actually present initially. To handle the case of $\mathbf{x} = \mathbf{0}$, start with initial context of some positive amount of a voter V that votes yes or not based on whether $\phi(\mathbf{0}) = 1$, and all other species X_U consume V via $X_U + V \to X_U$. ◀

B.3 Negative result: All stably decidable predicates are detection predicates

In this section we show that, unlike the case of computing functions $f : \mathbb{R}^k \to \mathbb{R}$ with numerical output (the focus of [20]), CRNs stably computing *predicates* are much more severely limited in computational power. We prove this using the fact, proven in [20], that such functions f must be *positive-continuous*, meaning that discontinuities can only occur when some input coordinate $\mathbf{x}(i)$ goes from 0 to positive. (Note the conceptual similarity to detection predicates, which can change their output only when some input coordinate $\mathbf{x}(i)$ goes from 0 to positive.) We connect this to predicates by showing that any CRN $\mathcal{C}$ stably computing a predicate ϕ can be augmented to stably compute a function σ_ϕ that is continuous only if the predicate stably computed by $\mathcal{C}$ is a detection predicate, proving that since such functions σ_ϕ must be continuous, then ϕ must be a detection predicate.

We recall the definition of a positive-continuous function from [20]. Intuitively, a positive-continuous function is only allowed to have discontinuities whenever some input goes from 0 to positive. For example, the function $f(x_1, x_2) = x_1$ if $x_2 > 0$ and $f(x_1, x_2) = 0$ otherwise is positive continuous though not continuous.

▶ **Definition B.9.** *A function $f : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ is* positive-continuous*, for all $U \subseteq \{1, \ldots, k\}$, f is continuous on the domain $D_U = \left\{\mathbf{x} \in \mathbb{R}^k_{\geq 0} \mid \mathbf{x}(i) > 0 \iff i \in U\right\}$.*

To understand the definition, it helps to understand first what the domains D_U look like. In the case $k = 2$. The domains D_U are the origin, the positive x and y axes, and the set $\{(x, y) \mid x, y > 0\}$. If $U = \{1\}$, then D_U is the positive x axis as the vectors $\mathbf{x} \in D_U$ must satisfy $\mathbf{x}(1) > 0$ and $\mathbf{x}(2) = 0$. A positive continuous function is one which must be continuous inside each domain, but is allowed discontinuities as it moves across boundaries. If useful, a positive continuous function can be understood as a collection of 2^k continuous functions, each defined on a different piece of the positive orthant.

Given a predicate $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$, define the *sum characteristic function* of ϕ, $\sigma_\phi : \mathbb{R}^k_{\geq 0} \to \mathbb{R}_{\geq 0}$ for all $\mathbf{x} \in \mathbb{R}^k_{\geq 0}$ by

$$\sigma_\phi(\mathbf{x}) = \begin{cases} \sum_{i=1}^k \mathbf{x}(i) = \|\mathbf{x}\|_1 & \text{if } \phi(\mathbf{x}) = 1 \\ 0 & \text{if } \phi(\mathbf{x}) = 0 \end{cases}$$

▶ **Lemma B.10.** *Let $\phi : \mathbb{R}^k_{\geq 0} \to \{0, 1\}$. If ϕ's sum characteristic function σ_ϕ is positive-continuous, then ϕ is a detection predicate.*

Proof. We show the contrapositive, that if ϕ is not a detection predicate, then the induced σ_ϕ is not positive-continuous. Suppose ϕ is not a detection predicate. Then, for some $U \subseteq \{1, \ldots, k\}$ the region D_U contains points $\mathbf{x}, \mathbf{y} \in D_U$ such that $\phi(\mathbf{x}) \neq \phi(\mathbf{y})$, as all detection predicates cannot change their output within a domain D_U. Note that this implies $D_U \neq \{\mathbf{0}\}$, i.e., $U \neq \emptyset$. Let ℓ denote the straight line connecting $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^k$. Such an ℓ is completely contained in D_U, as D_U is a convex set. Consider the image of ℓ under σ_ϕ, denoted $\sigma_\phi(\ell) \subseteq \mathbb{R}_{\geq 0}$. Since at least one point along ℓ does not satisfy ϕ, we have $0 \in \sigma_\phi(\ell)$. Furthermore, since $\mathbf{x}, \mathbf{y} \neq \mathbf{0}$, there is $\varepsilon > 0$ such that $\|\mathbf{x}\|_1 > \varepsilon$ and $\|\mathbf{y}\|_1 > \varepsilon$. Since $\|\cdot\|_1$

is a linear function, this implies that for all points $\mathbf{z} \in \ell$, $\|\mathbf{z}\|_1 > \varepsilon$. Thus, all non-zero elements of $\sigma_\phi(\ell)$ are greater than ε. This implies 0 is an isolated point of $\sigma_\phi(\ell)$. Since $\sigma_\phi(\ell)$ contains an isolated point, it is not a connected set. As a continuous function must preserve connectedness, this implies that σ_ϕ is not continuous on D_U, i.e., not positive continuous. ◀

▶ **Lemma B.11.** *If $\phi : \mathbb{R}^k_{\geq 0} \to \{0,1\}$ is stably decidable by a CRD, then ϕ is a detection predicate.*

Proof. Let $\mathcal{D}$ be a CRD stably deciding ϕ. We show how to convert $\mathcal{D}$ into a CRC $\mathcal{C}$ that stably computes the sum characteristic function σ_ϕ. Since all functions stably computable by a CRC are positive-continuous [20], Lemma B.10 implies ϕ is a detection predicate.

For each input species X_i, add the reaction $X_i \to X_i' + Y$, where X_i' is the i'th input species to $\mathcal{D}$. For each yes-voter T in $\mathcal{D}$, add the reaction $T + \hat{Y} \to T + Y$, and for each no-voter F, add the reaction $F + Y \to F + \hat{Y}$.

Let $\mathbf{c}$ be a state reached from the initial state $\mathbf{x}$. If any of the input species X_i are present in $\mathbf{c}$, we apply the reaction $X_i \to X_i' + Y$ until all these species are gone from the state. Once this is done, the concentrations of Y and $\hat{Y}$ satisfy $[Y] + [\hat{Y}] = \sum_{i=1}^k \mathbf{x}_i$. Denote this reached state by $\mathbf{c}'$. Since $\mathcal{D}$ stably decides ϕ, there is a stable state $\mathbf{o}$ reachable from $\mathbf{c}'$ in which all species present are yes-voters if $\phi(\mathbf{x}) = 1$ and no-voters otherwise. In the former case, we may apply the reaction $T + \hat{Y} \to T + Y$ to convert all $\hat{Y}$ to Y. As no voters are present in the state, it is stable. Furthermore, the concentration of $\hat{Y}$ is exactly $\|\mathbf{x}\|_1$. If $\phi(\mathbf{x}) = 0$, then only no voters are present so running the reaction $F + Y \to F + \hat{Y}$ eventually will remove all Y, stabilizing on the correct output of 0. ◀

Two for One, One for All: Deterministic LDC–Based Robust Computation in Congested Clique

Keren Censor-Hillel
Technion, Haifa, Israel

Orr Fischer
Bar-Ilan University, Ramat Gan, Israel

Ran Gelles
Bar-Ilan University, Ramat Gan, Israel

Pedro Soto
Virginia Tech, Blacksburg, VA, USA

Abstract

We design a deterministic compiler that makes any computation in the Congested Clique model robust to a constant fraction $\alpha < 1$ of adversarial crash faults. In particular, we show how a network of n nodes can compute any circuit of depth d, width ω, and gate total fan Δ, in $d \cdot \lceil \frac{\omega}{n^2} + \frac{\Delta}{n} \rceil \cdot 2^{O(\sqrt{\log n \log\log n})}$ rounds in such a faulty model. As a corollary, any T-round Congested Clique algorithm can be compiled into an algorithm that completes in $T^2 n^{o(1)}$ rounds in this model.

Our compiler obtains resilience to node crashes by coding information across the network, and its main underlying observation is that we can leverage locally-decodable codes (LDCs) to maintain a low complexity overhead, as these allow recovering the information needed at each computational step by querying only small parts of the codeword, instead of retrieving the entire coded message, which is inherent when using block codes.

The main technical contribution is that because erasures occur in known locations, which correspond to crashed nodes, we can *derandomize* classical LDC constructions by deterministically selecting query sets that avoid sufficiently many erasures. Moreover, when decoding multiple codewords in parallel, our derandomization *load-balances* the queries per-node, thereby preventing congestion and maintaining a low round complexity.

Deterministic decoding of LDCs presents a new challenge: the adversary can target precisely the (few) nodes that are queried for decoding a certain codeword. We overcome this issue via an adaptive doubling strategy: if a decoding attempt for a codeword fails, the node doubles the number of its decoding attempts. We employ a similar doubling technique when the adversary crashes the decoding node itself, replacing it dynamically with two other non-crashed nodes. By carefully combining these two doubling processes, we overcome the challenges posed by the combination of a deterministic LDC with a worst case pattern of crashes.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Congested Clique, Fault Tolerance, Error Correction Codes

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.20

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2508.08740 [8]

Funding *Keren Censor-Hillel*: is supported in part by the Israel Science Foundation, grant 529/23. *Orr Fischer*: is supported in part by the Israel Science Foundation, grant No. 1042/22 and 800/22. *Ran Gelles*: supported in part by the United States – Israel Binational Science Foundation (BSF), grant No. 2020277.

Acknowledgements We would like to thank Merav Parter and Noga Ron-Zewi for helpful discussions.



© Keren Censor-Hillel, Orr Fischer, Ran Gelles, and Pedro Soto;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Robustness is a crucial component in the design of distributed algorithms, as faults lie at the heart of distributed computing environments. Thus, addressing various types of failures has been heavily studied, see, e.g., seminal results covered in classic books on distributed computing [2, 37, 44]. In this paper, we focus on *node crashes* in the Congested Clique model (introduced by Lotker et al. [36]), in which the computing devices use bandwidth-restricted point-to-point communication over a complete network graph. So far, despite abundant research in this model (see related work in Section 1.2), relatively less attention was given to fault tolerance questions in this useful model [3, 15, 24, 33, 34].

In some settings, such as the Congest model or the work in Congested Clique of [33, 34], if node crashes are allowed then the requirement is that the output of the computation corresponds only to the inputs of non-crashed nodes. In contrast, Censor-Hillel and Soto [13] show how to avoid losing any information in the Congested Clique despite node crashes. They do this by having each node encode its input and the results of any subsequent local computations using erasure correction codes, split the codewords to pieces and distribute them to the nodes of the network. Upon a node crash, the other nodes collect the pieces of the relevant codeword and decode it in order to continue the computation. This approach implies that any task over a network of n nodes (with the typical setup of $O(n \log n)$ bits of input per node which we will also work with here) can be computed in $O(n)$ rounds despite crashes: after each node distributes its inputs in an encoded way, all other nodes gather all the pieces and reconstruct the entire $O(n^2 \log n)$-bit input, from which they can locally compute the output. The work [13] beats this bound for certain tasks whose circuit representation has good properties. While their work can achieve fast resilient algorithms for well-behaved circuits, e.g., retain the $\tilde{O}(n^{1/3})$-round complexity for matrix multiplication [12] even with faults, for a *general circuit*, their resilient algorithm may incur a multiplicative overhead of n rounds, which is worse than the solution that simply learns the entire encoded input.

In this paper, we provide a compiler that makes Congested Clique algorithms robust to a constant fraction of node crashes, by computing general circuits faster. This results in a compiler with a complexity of $T^2 n^{o(1)}$ rounds for a Congested Clique algorithm of T rounds, and thus it beats the $O(n)$-round solution when $T = o(n^{1/2-o(1)})$. In the crash model we consider, node crashes occur at the start of a round, and the adversary may crash up to αn nodes in total throughout the algorithm, for a fault parameter $\alpha \in [0, 1)$.

▶ **Theorem 1.** *Let* ALG *be any Congested Clique algorithm, such that each node's input and randomness string is of size at most $\widetilde{O}(n)$ bits, and that completes in T rounds. Then, for any $\alpha \in [0, 1)$, there is an equivalent algorithm* ALG′ *that is resilient to an α fraction of crashes and completes in $T^2 n^{o(1)}$ rounds.*

A concrete example for an application of Theorem 1 is computing exact single-source-shortest-paths (SSSP) in weighted undirected graphs. The $\tilde{O}(n^{1/6})$-round algorithm of [6] translates by our compiler to an algorithm that completes in $n^{1/3+o(1)}$ rounds, even if any αn nodes may crash during its execution, for any $\alpha < 1$.

At the heart of our compiler is a faster deterministic algorithm for computing general circuits in the faulty model. The following is our main technical contribution.

▶ **Theorem 2.** *Let C be circuit of depth d, max total-fan Δ, and width ω. Then, for any $\alpha \in [0, 1)$ there exists a deterministic Congested Clique algorithm for computing the output of C in the presence of αn crashes, whose round complexity is $d \cdot \lceil \frac{\omega}{n^2} + \frac{\Delta}{n} \rceil \cdot 2^{O(\sqrt{\log n} \log \log n)}$.*

While our transformation also goes through computing circuits, our algorithm differs from that of [13] in two main aspects, which we discuss next.

The first regards the initialization phase of the resilient algorithm. Note that if the adversary fails even a single node before the start of the computation, then this node's input is lost forever. The solution taken by [13] is promising $1/(1-\alpha)$ *quiet* rounds where no crashes can happen. These quiet rounds can be used to encode and distribute the nodes inputs. We take a different approach: we consider the inputs at the start of the computation as *encoded versions* of the inputs to the Congested Clique model (similar to Spielman's coded computation [45]). Since our robust algorithm is such that the outputs are also encoded in this manner, we get *composability* for free – we can simply start another computation after completing a former one.

Second, instead of using block error correction codes, we use *locally decodable codes (LDCs)* [31, 46], which allow a node to query only a small number of other nodes in order to decode only the information it needs for its next local computation. This yields a dramatic improvement in the round complexity of computing a circuit because of its huge effect on congestion. Working with LDCs gives rise to new challenges, because decoding an LDC codeword can be easily targeted by the adversary which can crash so many nodes. The crux of the proof of Theorem 2 shows how to overcome this, and that in fact it can be implemented in a *deterministic* manner.

Theorem 1 is immediately established by combining our robust computation of circuits in Theorem 2 with the natural conversion of any Congested Clique algorithm to a circuit (see a formal proof in, e.g., [13]).

▶ **Lemma 3.** *Let* ALG *be a Congested Clique algorithm that computes a function f in T rounds, where the node's input and randomness strings are of size $\widetilde{O}(n)$. Then, there is a circuit with depth $d = 2T+1$, width $\omega = \widetilde{\Theta}(Tn^2)$, and maximal gate total fan of $\Delta = \widetilde{O}(Tn)$, whose inputs are all the $\widetilde{O}(n^2)$ input bits of the nodes and whose outputs are the outputs of the nodes after running* ALG.

1.1 Technical Overview

Our main theorem states that we can robustly compute a circuit C of depth d and arbitrary gates, despite a possible (worst-case) crash of an α fraction of the nodes. Towards this end, the network computes the gates of C, layer by layer, where each node is assigned some of the gates in each layer in a dynamic manner that adapts to node failures.

As mentioned above, when a node crashes, it can no longer send messages, and thus any information that was privately held in that node's memory is lost forever. To be resilient to crashes and avoid losing important information, we need to store information "in the network" so that it can be retrieved despite a constant fraction of node crashes. Indeed, [13] used block error correction codes (ECCs) to encode data and split the resulting codewords across the network. This way, each piece of information can be retrieved by querying all the nodes for their part of the codeword; the decoding succeeds even if a constant fraction of the nodes crash, where the constant depends on the strength of the ECC in use. The drawback is that even if a single bit of information is needed from a coded string, its entire codeword needs to be decoded.

Our starting point is that we replace block codes with Locally Decodable Codes (LDCs). Informally, such codes allow decoding specific *parts* of the message, rather than decoding the entire message. Furthermore, decoding does not require obtaining the entire (corrupted) codeword, but rather queries relatively small parts of it (a subpolynomial number of symbols), while still guaranteeing correct decoding of the desired symbol with a high probability.

Hence, switching to LDCs benefits our algorithm in the sense that nodes make *less queries* in order to retrieve *exactly* the information they need for the computation. The advantage of LDCs over standard (i.e., block-codes) ECCs becomes clear when considering the case where some node is assigned to compute a gate with n inputs, each of which is stored in a different codeword. With standard ECCs, this means that the node must access all n codewords and retrieve all information bits, i.e., n^2 bits, assuming n-bit codewords, which causes high congestion.

Algorithm Overview. The high-level idea of our robust circuit computation algorithm is as follows. Consider a circuit C, whose inputs are distributed over the network in an encoded manner using some LDC code. The network computes C layer by layer. That is, let $\mathsf{gates}(1)$ be the first layer of gates in C, i.e., all the gates whose inputs are the inputs of C. We first distribute the tasks of computing these gates across the (non-crashed) nodes so that each node is assigned roughly the same number of gates to compute. Each node then attempts to compute all the gates allocated to it. To do this, the node first obtains the inputs for each gate assigned to it by decoding the corresponding inputs of C, which are stored in the network using codewords of an LDC. If this information retrieval is successful, the node computes the outputs of the gates allocated to it, and then "stores" them in the network by encoding them with an LDC and distributing the codewords to the nodes in the network. Once the first layer is computed and stored in the network, the nodes continue to compute the second layer of gates in C, denoted $\mathsf{gates}(2)$, which includes all gates whose inputs are either the inputs of C or the outputs of the gates in the first layer, $\mathsf{gates}(1)$. This continues until all the outputs of C are computed and stored in the network. Naturally, crashes that occur during the algorithm may prevent the network from completing the computation of a specific layer and progressing to the next layer, which we discuss next.

Overcoming Crashes I. In our robust circuit computation algorithm, each gate is assigned to a dedicated node responsible for its computation. If that node crashes, the gates assigned to it remain uncomputed. A trivial solution is to reassign any uncomputed gate in the current layer of C (whose original node is crashed) to a new node that is not crashed. However, the adversary could then crash this new node and eventually cause a delay of αn rounds, which is extremely expensive. Our strategy is different: when a node crashes, we reassign its gate(s) to *two* fresh nodes. If both of those nodes crash, we again double the redundancy, forcing the adversary to double its effort to keep the gate uncomputed. After at most $\log n$ such iterations, every gate is guaranteed to be computed by at least one live node. This progressive doubling remains feasible without causing excessive congestion because of two factors: First, the nodes that do not crash successfully compute and store their assigned gates. These nodes are now available to take over the gates of the crashed nodes. Second, the adversary cannot (effectively) corrupt too many nodes in the same round: If the adversary crashes too many nodes during a short period of time, we call this step *overwhelmingly faulty*, and simply restart the computation of this layer with the remaining nodes. While this translates to no progress, it reduces the adversary's budget of crashes and hence cannot occur too many times.

Deterministic LDCs and Congestion. The decoding algorithm of LDCs is *inherently random*. Indeed, if a fixed (small) number of codeword symbols are queried during a decoding attempt and these symbols are corrupted, then decoding is certainly impossible. If so, the code cannot correct a constant fraction of corruptions, as would be normally expected. In particular, given a budget of αn node crashes, an all-knowledgeable adversary may be able to crash a subset of nodes in a way that prevents any meaningful progress of the deterministic computation.

Despite the above conundrum, our robust algorithm is fully deterministic. In particular, we derandomize the LDC decodings performed throughout the computation while maintaining resilience to αn node crashes. Our derandomization relies on two important properties, specific to our model. First, when a node crashes, all other nodes are aware of this event because the crashed node does not send any messages from the round in which it crashed. This allows the remaining nodes to maintain a consistent view of the crashed and alive set of nodes. Second, crashed nodes that are queried for their respective parts of the LDC codeword do not reply, and thus the LDC decoding algorithm is missing some parts of the queried codeword, known as *erasure* corruptions. These are easier to correct than when the codeword contains incorrect information. The combination of erasure corruptions and knowledge of which nodes have crashed in each round allows a decoder to predict whether a specific set of queries will result in successful decoding. Thus the decoder can pick a set of queries that is guaranteed to succeed if no further nodes crash.

While the above idea derandomizes the (inherently random) LDC decoding algorithm, it creates a new challenge regarding the resulting congestion. To illustrate this challenge, suppose that a node performs the above deterministic selection of queries *separately* for each piece of information it wants to retrieve. Then, it may end up querying the same subset of nodes over and over again, thus causing a large congestion. Randomized decoding averts this issue by querying a set of nodes in a near-uniform distribution. However, even if we could deterministically replicate this querying distribution, we face again the issue mentioned above, where many of these queries are erased and do not lead to a correct decoding.

Nevertheless, our analysis, which is based on the probabilistic method, shows that it is possible to select query sets for multiple (independent) LDC-decoding instances in the presence of a constant fraction of erasures in positions known to the decoding algorithm, so that the following hold simultaneously: (i) each decoding instance successfully decodes the correct information, and (ii) the congestion per queried node is small, i.e., the queries are well-distributed over the network. To show the latter, we analyze an equivalent bins-into-balls experiment, showing that the event that too many balls (queries) aggregate in one specific bin (node) happens with small probability. Union-bounding over all nodes keeps the probability of the bad event below 1, thus proving the existence of good query sets that avoid congestion.

Overcoming Crashes II. The above discussion implies that the adversary *cannot* select a set of αn nodes to crash for invalidating many of the LDC decoding attempts throughout the computation, as long as these indices are known to the nodes. However, the adversary may decide to crash nodes *after* a decoder fixes its selection of nodes to query in a given round, as this selection depends only on nodes that are crashed *prior* to that round. To overcome this problem, the nodes dynamically increase the number of times they attempt to LDC decode each piece of information, according to corruptions made so far. Namely, if the decoding of some LDC-encoded information fails due to *new* corruptions of the queried nodes, then the decoding node performs *two* independent decoding attempts. These new queries depend on all the crashes so far, and in particular, on the "new" crashes that invalidated the original decoding attempt. If these two attempts fail as well (due to new crashes that occur after the nodes queried by these two attempts are decided), the decoding node doubles its number of attempts again, and so on. Overall, after a logarithmic number of doublings, this approach potentially causes a large, near-linear number of LDC decoding attempts, and the adversary can only fail a constant fraction of them without exceeding its budget. Note that it only takes one successful attempt to move on, so the adversary must fail all attempts of a single codeword to prevent progress.

Recap. We can now summarize the overview of our robust circuit computation. For a given circuit C, the computation goes layer by layer, where computing a layer of C means: (1) assigning uncomputed gates of the layer to non-crashed nodes; (2) retrieving the inputs to the gates of this layer, that are stored in the network via an LDC during the computation of previous layers; (3) computing the gates; (4) storing the outputs of the gates via an LDC. Technically speaking, this computation of each layer is done in two nested loops: The external loop doubles, in each iteration, the number of nodes responsible for computing some uncomputed gate (we call this loop *the* NodeDoubling *loop*). The internal loop doubles, in each iteration, the number of independent decoding attempts each node makes for each uncomputed gate assigned to it (we call this loop *the* AttemptDoubling *loop*). Section 3 fully details our algorithm. The algorithm's analysis and the derandomization of the LDC in our setup appear in the full version of the paper [8].

1.2 Additional Related Work

Since its introduction for faster MST computation [36], the Congested Clique model has been extensively explored during recent decades for various tasks. The MST complexity was eventually shown to be constant [40] following a beautiful line of work [26, 28, 30, 32]. Additional examples include routing [35], coloring [4, 14, 16, 17, 42, 43], subgraph finding [7, 9, 10, 19, 21, 29, 41], and many more. Hardness of obtaining lower bounds in this model is established in [20].

Fault-tolerance in the Congested Clique model was explored by [33,34] for graph realization problems under crash-faults, by [3, 15] for recognizing connectivity and hereditary properties under Byzantine faults, and by [23, 24] for general computations under edge faults. In particular, [24] employs LDCs as means to concentrate information from many nodes into few.

Coding theory, in various forms, has been extensively used in many other areas of distributed computing, including: distributed zero-knowledge [5, 27], proof labeling schemes [11, 22], the beeping model [1, 18, 25], and distributed interactive proofs [39].

2 Preliminaries

For an integer $n \geq 1$ we denote $[n] = \{1, 2, \ldots, n\}$. All logarithms are taken to base 2 unless otherwise mentioned. We say that an event occurs with high probability (in n, which is usually implicit) if its probability is at least $1 - 1/n^{10}$. For a string x and for any $i \in [|x|]$, let $x[i]$ denote the i-th symbol of x.

2.1 Computation Model

Suppose a Congested Clique network, where n nodes, $v_1, \ldots, v_n$, communicate in synchronous rounds by exchanging $b \log n$-bit messages in an all-to-all fashion, for some constant $b \in \mathbb{N}$. Throughout the computation, an adversary may choose to crash up to αn nodes, where the constant $\alpha \in [0, 1)$ is a parameter of the model. A crashed node does not send any messages starting from the round in which it is crashed. The corruption is *worst case*: an all-knowledgeable adversary bases its decision on which nodes to fail on all its available information, including the algorithm that the nodes execute, their inputs, and their local randomness (if any). Note that all nodes know which nodes are non-faulty at the end of each round, denoted as the set $\mathcal{A}$ (to indicate that they are *alive*).

The compiler of Algorithm 5 is completely deterministic, in the sense that it does not add any new randomness. In particular, The robust algorithm ALG′ remains deterministic if ALG was deterministic, and similarly, it is randomized if ALG was so. In the latter case, we assume that the randomness of ALG is given to the representing circuit C as input.

Coded inputs and outputs. In the Congested Clique model, it is common to refer to problems in which each node holds a private input x of $O(n \log n)$ bits. As explained in the introduction, in our faulty model, the inputs must be coded in a way that prevents them from being lost if some node crashes before the first round of communication. We thus consider inputs as encoded via an LDC code (see Definition 4 in Section 2.3 below), and the respective codeword is distributed across the nodes of the network. We employ a (q, δ, ϵ)-LDC code with block length n, whose parameters will be specified later. Such a code guarantees that every bit of the input x (of each node) can be retrieved by querying q nodes with probability $1 - \epsilon$ over a uniform choice of the randomness string, even if δn of the nodes have crashed. It will be the case that $\alpha < \delta$.

We require the output to be stored in the network in a similar way: the outputs should be encoded via an LDC code whose resulting codewords are split among the nodes, so that it is possible to retrieve each bit of the output despite δn crashes. This choice allows the composition of computations, where the outputs of one computation are the inputs of the next.

2.2 Layered Circuits

We identify a circuit C with a directed acyclic graph $C = (V_C, E_C)$ in which every gate is associated with a node, and every wire connecting gates is associated with an edge. Each bit-input to the circuit (an input gate) is associated with a leaf node and each output of the circuit (an output gate) with a root node. Other nodes are associated with the computational gates of the circuit. We say that a node $g \in V_C$ *depends* on a node $g' \in V_C$ if there is a directed path from g' to g. The notion of gate dependencies induces *layers* in the circuit, where all input gates are in layer 0, and a gate g is placed in layer i if i is the minimal integer such that g depends only on nodes in layers at most $i - 1$. For a gate $g \in V_C$, we denote by $\mathsf{layer}(g)$ the layer of g, and let $\mathsf{gates}(i) = \{g \in V_C \mid \mathsf{layer}(g) = i\}$ be the set of all the gates in layer i. The depth of C, denoted $d = d(C)$, is defined to be $\max_{g \in V_C} \mathsf{layer}(g)$. We denote by $\mathsf{wires}(i) = \{(u, v) \in E_C \mid \mathsf{layer}(u) = i\}$ the set of all wires that go out of the gates in layer i. Note that $\mathsf{wires}(0)$ are the inputs to the circuit. For a gate g, denote by $\mathsf{fan\text{-}in}(g)$ its in-degree and by $\mathsf{fan\text{-}out}(g)$ its out-degree; let $\mathsf{fan}(g) = \mathsf{fan\text{-}in}(g) + \mathsf{fan\text{-}out}(g)$ be its *total fan*. The *width* of the circuit, $\omega = \omega(C)$, is defined as the maximal number of outgoing wires of any layer, $\omega = \max_i(\mathsf{wires}(i))$. We assume throughout that all parameters of the circuit are polynomial in the size of the network, i.e. $d, \omega, \Delta = O(\mathrm{poly}(n))$. This fits the case where C represents a Congested Clique algorithm with $O(n \log n)$ bits of input per node. We note, however, that the statement of Theorem 2 holds, up to logarithmic terms, for any parameters d, ω, and Δ.

2.3 Error Correcting Codes and Locally Decodable Codes

For an alphabet Σ, the Hamming distance of two strings $x, y \in (\Sigma \cup \bot)^*$ of the same length, i.e., $|x| = |y|$, is the number of indices for which x and y differ and is denoted by $\mathrm{Hamm}(x, y) = |\{i \mid x[i] \neq y[i]\}|$. For two strings $x \in \Sigma^*$, $y \in (\Sigma \cup \{\bot\})^*$ and value $c \in (0, 1)$, we say that y can be obtained by a c-fraction of erasures from x if $|x| = |y|$, and for all $i \in [|x|]$, it holds that either $x[i] = y[i]$ or $y[i] = \bot$, where the latter case happens at most $c|x|$ times. An index i in which $y[i] = \bot$ is called an erasure.

For a prime power p, we denote by $\mathbb{F}_p$ the finite field of size p. An *error correcting code* is a mapping $\mathsf{Enc} : \mathbb{F}_p^K \to \mathbb{F}_p^N$ that takes K symbols of the alphabet $\mathbb{F}_p$ into N symbols of the alphabet $\mathbb{F}_p$.[1] The value N is called the block length of the code. The ratio K/N is called

[1] We can map $[p]$ and $\mathbb{F}_p$ with a fixed isomorphism, so that $\mathsf{Enc} : [p]^K \to [p]^N$.

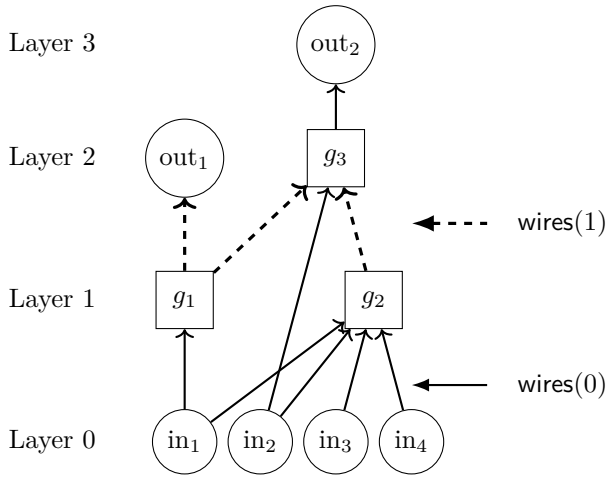


Figure 1 An example of a circuit C of depth $d = 3$ and width $\omega = 6$. We have $\mathsf{gates}(1) = \{g_1, g_2\}$ and $\mathsf{gates}(2) = \{g_3\}$. The gate g_2 has $\mathsf{fan\text{-}in} = 4$ and $\mathsf{fan\text{-}out} = 1$ giving $\mathsf{fan} = 5$, while the gate in_1 has $\mathsf{fan\text{-}out} = 2$ and $\mathsf{fan} = 2$. The set $\mathsf{wires}(0)$ and $\mathsf{wires}(1)$ are indicated on the figure.

the *rate* of the code. The *relative distance* of a code is the normalized Hamming distance between any two codewords, denoted $\delta = \min_{m \neq m'} \frac{1}{N}\mathrm{Hamm}(\mathsf{Enc}(m), \mathsf{Enc}(m'))$.

Next, we formally define the notion of locally decodable codes. For the purposes of our work, we only consider the *erasure* setting, in which we are given access to a possibly corrupted codeword y, obtained by erasing at most δ-fraction of some $\mathsf{Enc}(x)$ for some $x \in \mathbb{F}_p^K$, and an index $i \in [K]$, and our goal is to find the i-th symbol of x.

▶ **Definition 4** (Locally Decodable Codes (LDCs) for erasures). *An error correcting code* $\mathsf{Enc} : \mathbb{F}_p^K \to \mathbb{F}_p^N$ *is said to be a* (q, δ, ϵ)*-LDC if there exists a randomized decoding algorithm* Dec *that receives as input a string* $y \in (\mathbb{F}_p \cup \{\perp\})^N$ *and an index* $i \in [K]$*, performs at most* q *queries to* y*, and outputs a value with the following guarantee: if there exists* $x \in \mathbb{F}_p^K$ *such that* y *can be obtained by at most a* δ*-fraction of erasures from* $\mathsf{Enc}(x)$*, then* $\Pr(\mathsf{Dec}(y, i) = x[i]) \geq 1 - \epsilon$.

▶ **Definition 5** (Non-adaptiveness). *An LDC is called* non-adaptive *if for every call to its decoding algorithm* Dec*, the set of queries it performs given input index* i *is only a function of the randomness and the index* i*. In particular, a query does not depend on the outcome of previous queries.*

We can think of the decoding algorithm of a non-adaptive LDC code $\mathsf{Dec}(\cdot, i)$ as an algorithm with oracle access to the codeword, that first generates q indices to query, and then, once provided these (possibly corrupt) q symbols, returns the decoded message symbol.

The following smoothness property of LDCs means that decoding an index requires querying the codeword in a "smooth" (near-uniform) way. This property is important in order to avoid congestion when a node decodes multiple values.

▶ **Definition 6** (Smoothness). *An LDC is called* s-smooth *if there exists a decoding algorithm* Dec*, such that during any call to* Dec*, any entry* $j \in [N]$ *of the codeword is queried with probability at most* s*.*

The following theorem suggests that smoothness is an inherent property of LDCs, since any decoding algorithm can be transformed into a smooth one.

▶ **Theorem 7** ([31, Theorem 1]). *Every* (q, δ, ϵ)*-LDC of block length* N *is* $q/\delta N$*-smooth.*

3 Computing a Circuit in the presence of crashes

We show how to efficiently and deterministically compute a specified circuit C in the Congested Clique model, in the presence of up to αn crashes. We start, in Section 3.1, by describing the procedures Store, Retrieve, and BulkRetrieve used to store and retrieve information in our algorithm. In section Section 3.2 we describe another procedure, Allocate, that assigns gates to nodes in a balanced-manner. Finally, in Section 3.3, we describe our circuit computation algorithm based on these procedures.

3.1 The Store, Retrieve, and BulkRetrieve Procedures

As mentioned above, any information that may get lost due to node crash is stored in the network via an LDC. We first describe the LDC we use and then detail the store and retrieve procedures.

The LDC instantiation

We assume a fixed LDC code, whose exact details appear in the full version of the paper [8]. Specifically, for some power of prime $q = 2^{O(\sqrt{\log n})}$ and ρ such that $\rho^{-1} = 2^{O(\sqrt{\log n}\log\log n)}$, our LDC has an encoding function $\mathsf{Enc} : [q]^{\rho n} \to [q]^n$ and a decoding function Dec which, given an input index $i \in [n]$, smoothly queries q indices of the codeword and decodes correctly even if up to δ-fraction of the q queried symbols are erased, for some predetermined constant distance $\alpha < \delta < 1$. We assume that $n = q^r$ for some integer r (specified in the detailed construction); this assumption can be lifted using standard methods. Note that Definition 4 says that Dec is randomized, but our goal is to compute the circuit deterministically. We thus treat any randomness used by Dec as originating from some *randomness string*, but our implementation of obtaining such randomness strings will be *deterministic* rather than randomized which will render our implementation of Dec, and hence our circuit computation algorithm, deterministic.

The Store Procedure

The Store procedure "saves" information in the network in a robust way, by encoding it with an LDC and distributing the codeword among the nodes of the network.

Each node v_j begins the Store procedure with a bit-string U_j it wishes to store in the network. It first splits U_j into parts of size $\rho n \lfloor \log q \rfloor$ bits each (so that each can be represented by a string of ρn symbols over the alphabet $[q]$), padding with zeros as necessary. Set $\mathsf{last}_j = \lceil |U_j| / (\rho n \lfloor \log q \rfloor) \rceil$ and denote these parts $U_j^1, \ldots, U_j^{\mathsf{last}_j}$. The node v_j then encodes each U_j^i using Enc to obtain a codeword $\mathsf{Enc}(U_j^i) = L_j^i$ of size $|L_j^i| = n$ symbols.

Next, v_j distributes the codewords to the network nodes. Specifically, in round $i = 1, \ldots, \mathsf{last}_j$, it sends the symbols of L_j^i – one symbol to each node in the network. This takes a single round of communication because each symbol in the LDC codeword comes from the alphabet $[q]$ with $q = 2^{O(\sqrt{\log n})}$, hence it can be encoded in $\log q = O(\log n)$ bits. The formal description is depicted in Algorithm 1.

We say that v_j *stored* U_j in the network if v_j completed the Store procedure without crashing. The following is straightforward from Algorithm 1.

▶ **Observation 8.** *Storing a string U_j takes $O(\lceil |U_j| / (\rho n \lfloor \log q \rfloor) \rceil)$ rounds of communication.*

Algorithm 1 Store (for node v_j).

Input: A bit-string U_j.

1: Partition U_j into consecutive parts of size $\rho n \lfloor \log q \rfloor$ bits, padding the last part if necessary; denote these substrings $U_j^1, \ldots, U_j^{\mathsf{last}_j}$. ▷ $\mathsf{last}_j = \lceil |U_j| / (\rho n \lfloor \log q \rfloor) \rceil$
2: **for** $i = 1, \ldots, \mathsf{last}_j$ **do**
3: $L_j^i \leftarrow \mathsf{Enc}(U_i^j)$.
4: For all $t \in [n]$ in parallel, send $L_j^i[t]$ to v_t.
5: **end for**

The Retrieve and BulkRetrieve Procedures

In the Retrieve procedure, a node v_j is given as input an index w of some string U that was previously stored in the network using the Store procedure. Node v_j is additionally given a string R called the randomness string. One can think of this procedure as randomized, with R as its randomness, but in all our invocations of Retrieve, the string R is set deterministically, as will be explained later.

The goal of the Retrieve procedure is to retrieve the value of the w-th bit of the previously stored U. To that end, v_j first identifies the codeword that contains the bit w: recall that the Store procedure splits U into parts of size $\approx \rho n \log q$ bits. Denote by U^i the respective part and by i' the index of the symbol in U^i that contains the bit-value $U[w]$ in which we are interested. In the following, we say "decode $U[w]$" to actually mean decoding the respective index i' of the possibly corrupted codeword $\mathsf{Enc}(U^i)$ that contains the respective value.

To retrieve the value of $U[w]$ from the (stored) codeword $\mathsf{Enc}(U^i)$, the node v_j executes $\mathsf{Dec}(.)$ using the randomness string R and obtains indices of random q symbols of $\mathsf{Enc}(U^i)$ needed for the decoding. It is possible to learn these q indices in advance because the LDC is non-adaptive (Definition 5). The node v_j then queries the respective nodes for their stored symbols and provides $\mathsf{Dec}(.)$ with their replies. Note that crashed nodes do not reply, which translates to erasures given to $\mathsf{Dec}(.)$. Further, in hindsight, the randomness string R will be derandomized, which has the effect that all nodes know R. It will follow that v_j does not actually need to send any message in order to query any node; the q nodes will know that they are the nodes that should give information back to v_j, since they will know the identity of i', U^i and the value of R to begin with. See Lemma 14.

Under some circumstances, we allow the retrieval to fail, in which case the output is $\perp$. This could happen in two cases: (i) when there are too many erasures and $\mathsf{Dec}(.)$ returns $\perp$, or (ii) if one of the nodes queried during this Retrieve invocation crashes during the execution of this Retrieve invocation. For the former, our derandomization will guarantee that this event cannot happen if at most αn nodes have crashed. For the latter, in this case we set the output to be $\perp$ even if the decoding Dec successfully retrieves the symbol. This decision does not affect the correctness of the algorithm, but rather simplifies its analysis.

The BulkRetrieve procedure generalizes the above to allow retrieving multiple previously stored bits. Now, v_j is given as input a *collection* of indices W_j, where each index $w \in W_j$ refers to some (predetermined) $U_{(w)}$ that was previously stored in the network via a Store. The strings $U_{(w)}$ may be different for different values of w, or they may be the same. Additionally, the procedure gets a multiplicity parameter ℓ. The goal is to output the value of the bit in index w of $U_{(w)}$, for each index $w \in W_j$.

Algorithm 2 Retrieve (for node v_j).

Inputs: An index w to some string U previously stored in the network via the Store procedure and a randomness string R.

1: Identify the part U^i used by Store to encode $U[w]$ and the respective index i' in it that contains that value, i.e., the symbol $U^i[i']$ contains the bit $U[w]$.
2: Execute $\mathsf{Dec}(\cdot, i')$ with randomness string R, to obtain the q indices needed to decode $U^i[i']$. Set $S \subset V$ to be the nodes that hold these respective symbols.
3: Query the respective nodes in S. Treat symbols associated with crashed nodes as erasures.

Output:
4: If some node $v \in S$ crashes **during the execution of Line 3**, output $\perp$.
5: Otherwise, output the bit $U[w]$ contained in $\mathsf{Dec}(\mathsf{Enc}(U^i, i'))$ by providing the q replies (including erasures) when Dec queries the codeword $\mathsf{Enc}(U^i)$.

Towards this goal, all nodes in the network first *deterministically* compute a set of randomness strings $\mathcal{R}(v_j) = \{R_{v_j,w,i} \mid i \in [2^\ell], w \in W_j\}$ for each $v_j \in V$, with *good* properties, which we define and discuss in detail later in the section (see Definition 9). The deterministic generation of these strings is given by Lemma 10. After this step, all nodes $v \in V$ know $\mathcal{R}(v_j)$ for every v_j.

Next, v_j performs, for each index $w \in W_j$, a batch of 2^ℓ Retrieve procedures, where the i-th invocation uses randomness string $R_{v_j,w,i}$. Similar to the case of Retrieve, we allow some retrieves to fail and output $\perp$. If at least one of the 2^ℓ invocations of $\mathsf{Retrieve}(w, R_{v_j,w,i})$ succeeds, its output becomes the output of BulkRetrieve for the index w; otherwise, the respective output is $\perp$.

All the $|W_j| \cdot 2^\ell$ Retrieve invocations are executed in parallel. However, in order to avoid congestion, v_j pipelines requests targeted to the same node. That is, it sends at most one query to any node in any given round. Similar to above, the set of strings $\{U_{(w)}\}_{w \in W_j}$ is predetermined and known to all nodes, and the identities of the q nodes that are queried in a specific $\mathsf{Retrieve}(w, R_{v_j,w,i})$ are generated using $R_{v_j,w,i}$, which is also known to all nodes. Hence, each queried node can infer the respective $U_{(w)}$ for each query, without the need for v_j to communicate this data. The formal procedure is depicted in Algorithm 3.

Consider a specific instance of $\mathsf{BulkRetrieve}(W_j, \ell)$. The selection of randomness strings $\mathcal{R}(v_j)$ that v_j uses has a tremendous effect on the induced congestion. Indeed, assume that $\mathcal{R}(v_j)$ is such that *all* Retrieves query some v_i, implying $2^\ell |W_j|$ rounds of communication where in each round a single LDC symbol is communicated. This should be contrasted with the fully randomized case, where each node is queried in a near-uniform distribution (implied by the smoothness of LDC codes, Theorem 7), implying that each v_i is queried $2^\ell |W_j| \cdot q/n$ times, in expectation. Standard tail bounds show that the number of queries of the maximal node (and hence the round complexity) is bounded by $2^\ell |W_j| \cdot q/n \cdot O(\log n)$. This gives that there *exists* a way to select the randomness strings $\mathcal{R}(v_j)$ while maintaining the same round complexity.

However, while the above gives uniform query locations for the goal of controlling congestion, it does not address the problem that many locations might be erased. To illustrate this point, assume that out of the 2^ℓ Retrieve instances, all but one are querying mostly erased symbols (crashed nodes), and only one Retrieve correctly decodes the value. The output of BulkRetrieve would be correct in this case, but in this scenario the adversary needs to crash only a single additional node in order to fail it.

Algorithm 3 BulkRetrieve (for node v_j).

Inputs: An ordered collection W_j of indices, where each $w \in W_j$ refers to a predetermined string $U_{(w)}$, that was previously stored by some node executing Store. A multiplicity parameter ℓ.

1: Compute a collection of randomness strings $\mathcal{R}(v_j) = \{R_{v_j,w,i} \mid w \in W_j, i \in [2^\ell]\}$ using Lemma 10.
2: **parallel for** each $w \in W_j$ **do**
3: **parallel for** $i = 1, \ldots, 2^\ell$ times **do**
4: Run Retrieve$(w, R_{v_j,w,i})$. If v_j needs to query the same node multiple times in all of these parallel instances, send at most one query per round, until all queries are sent.
5: **end parallel for**
6: **end parallel for**

7: **Output:** For each w, the output is the output of the first successful Retrieve$(w, R_{v_j,w,i})$, if any, or $\perp$ otherwise.

Indeed, what we show is even stronger than mimicking a fully randomized case by some naïve load balancing. The following Lemma 10 shows that we can find a set of randomness strings that maintains a similar round complexity even if each codeword has αn symbols erased, and furthermore, *each individual* Retrieve *succeeds decoding the respective value*, as long as no new crashes happened during that Retrieve. In other words, we can de-randomize the random sampling of codeword symbols to query while (i) maintaining complexity (by controlling congestion) and (ii) performing only "useful" queries, hence maintaining our resilience to an all-knowledgeable adversary. Our choice of randomness strings guarantees that the adversary must waste 2^ℓ of its crashing budget in order to fail the BulkRetrieve, which is crucial for the correctness proof.

We now define the notion of good randomness strings, namely, strings that provide the above properties for the BulkRetrieve procedure.

▶ **Definition 9** (Good randomness strings). *Fix a node v_j, parameters W_j, ℓ, and the set of non-crashed nodes $\mathcal{A}$. A collection of randomness strings $\mathcal{R}(v_j) = \{R_{v_j,w,i}\}_{w \in W_j, i \in [2^\ell]}$ is called* good *for an instance of* BulkRetrieve(W_j, ℓ)*, if the following holds:*

(1) *Each node is queried at most $O(\lceil \frac{2^\ell |W_j| q}{n} \rceil \log n)$ times in total by v_j, and*

(2) *For all $w \in W_j$ and $i \in [2^\ell]$, the invocation of* Retrieve$(w, R_{v_j,w,i})$ *succeeds (given no further changes in $\mathcal{A}$).*

In the full version of the paper [8], we prove the following.

▶ **Lemma 10.** *There is a zero-round deterministic algorithm which, given the set of non-crashed nodes $\mathcal{A}$, a collection of indices W_j, and a multiplicity parameter ℓ, computes a good collection of randomness strings $\mathcal{R}(v_j)$ for v_j.*

With the above, the following is immediate.

▶ **Lemma 11.** BulkRetrieve(W_j, ℓ) *takes $O(\lceil \frac{2^\ell |W_j| q}{n} \rceil \log n)$ rounds.*

3.2 The Allocate Procedure

The purpose of the Allocate procedure is to assign a set of given gates that need to be computed to non-crashed nodes. The procedure is deterministic and runs locally on each node, without any communication. However, all nodes reach the same allocation, since they all have the same knowledge regarding crashed nodes and regarding failed LDC queries, where the latter is due to the randomness strings being generated deterministically by all nodes and known to all (due to Lemma 10 above and the upcoming Lemma 14 which essentially says that the nodes are able to keep a consistent view of all of these variables by careful bookkeeping).

In more detail, the procedure is given as input the current set of non-crashed nodes $\mathcal{A}$, a set of gates $G \subseteq V_C$ (of the circuit $C = (V_C, E_C)$, known to all), and a multiplicity parameter ℓ_1. For each gate $g \in G$, Allocate assigns g to a set of $\min(2^{\ell_1}, |\mathcal{A}|)$ nodes from $\mathcal{A}$ using the following sequential "greedy" process: Sort the gates in G by their total fan (denoted fan), in descending order. Assign the gates one by one to a set of $\min(2^{\ell_1}, |\mathcal{A}|)$ distinct nodes in $\mathcal{A}$ whose loads are minimal (break symmetry by node IDs). The *load* of a node v, denoted $\lambda(v)$, is defined as the sum of the total fan of all gates assigned to it so far during this Allocate instance. See Algorithm 4.

Algorithm 4 Allocate (for node v_j).

Inputs: A set G of gates and a multiplicity parameter ℓ_1.

Let $g_1, \ldots, g_{|G|}$ be the gates of G, sorted in descending order of fan (break ties consistently).
Set $\lambda(v) \leftarrow 0$ for all nodes $v \in \mathcal{A}$.
$G_j \leftarrow \emptyset$.
$L \leftarrow \min(2^{\ell_1}, |\mathcal{A}|)$.
for $i = 1, \ldots, |G|$ **do**
    Let $u_1, \ldots, u_L$ be the L nodes in $\mathcal{A}$ with the minimal loads (break ties by IDs).
    for each node $u \in \{u_1, \ldots, u_L\}$ **do**
        Assign g_i to the node u.
        $\lambda(u) \leftarrow \lambda(u) + \mathsf{fan}(g_i)$.
        if $u = v_j$ **then** $G_j \leftarrow G_j \cup \{g_i\}$.
        end if
    end for
end for

Output: The set G_j of gates assigned to v_j.

The assignment can be computed locally in a consistent manner across all non-crashed nodes without any communication, since all relevant information, namely G, ℓ_1, and $\mathcal{A}$, is known to all nodes in $\mathcal{A}$. Note that since this is a local computation procedure, we can assume that set $\mathcal{A}$ does not change throughout the computation, and that all nodes use the same set $\mathcal{A}$ representing the non-crashed nodes at the beginning of that round.

The following lemma bounds the load assigned to each node, and is proven in Section B.

▶ **Lemma 12.** *Let* $L = \min(2^{\ell_1}, |\mathcal{A}|)$, $P = \sum_{g \in G} \mathsf{fan}(g)$, *and assume* $\max_{g \in G} \mathsf{fan}(g) \leq \Delta$. *Then,* Allocate$(G, \ell_1)$ *puts a maximal load of* $\max(4PL/|\mathcal{A}|, \Delta)$ *on each node.*

3.3 The Circuit Computation Algorithm

We can now complete the description of our circuit computation algorithm, presented in Algorithm 5. The algorithm takes as input a circuit C whose inputs (the wires $\mathsf{wires}(0)$) are already stored in the network.

The algorithm computes the gates of C layer by layer, in a sequence of d steps referred to as layer-*steps*. For layer-step $i = 1, \ldots, d$, we assume that $\mathsf{wires}(0), \ldots, \mathsf{wires}(i-1)$ have already been stored by previous iterations, and the goal is to compute and store $\mathsf{wires}(i)$. To this end, the nodes execute Allocate, which assigns to each non-crashed node v_j a set of gates $G_j \subseteq \mathsf{gates}(i)$ to compute and store their output wires (line 5).

Then, each node v_j tries to retrieve the input wires of the gates G_j assigned to it via the BulkRetrieve procedure (line 8). If successful, the node computes the gates assigned to it (line 12) and obtains the values of all output wires U_j of the gates G_j. The node then stores these wires in the network (line 13).

However, crashes that occur during this computation may hinder the computation of some wires. To overcome this issue, the computation of layer i consists of two nested loops. The outer loop, which we call the NodeDoubling-loop, iterates over $\ell_1 = 1, \ldots, \lceil \log n \rceil$ and doubles the number of nodes that try to compute a given gate. The inner loop, called the AttemptDoubling-loop, iterates over $\ell_2 = 1, \ldots, \lceil \log \Lambda \rceil$ for some parameter Λ (fixed in the analysis), and doubles the number of retrieval attempts a given node performs for each input wire assigned to it.

If during the computation of layer i, more than $c_f n/(q \log n)$ new crashes have occurred, for some sufficiently small constant $c_f > 0$ determined later, we re-start the computation of that layer with the remaining nodes. Namely, we maintain a counter $f_{i,rep}$ of newly crashed nodes in the layer-step, which is initialized at the start of the layer-step to be 0. Once it passes $c_f n/(q \log n)$, we reset the counter to 0, reset the multiplicity parameters ℓ_1, ℓ_2 to 1, and retry to compute and store all remaining unstored wires in $\mathsf{wires}(i)$. This action is captured in lines 16–20, assisted by the variable *rep*, that counts the number of repetition attempts of computing layer i. We call such a repetition of a layer-step *overwhelmingly faulty*:

▶ **Definition 13.** *Let $c_f > 0$ be a sufficiently small constant. A repetition of a* layer*-step is called overwhelmingly faulty if $c_f n/(q \log n)$ new crashes occur during this step.*

The next lemma captures the following observation: the nodes are capable of "bookkeeping" the progress of the computation at any given round of Algorithm 5. This bookkeeping information includes gates that were computed and stored, gates that still need to be computed, Retrieve calls that succeeded and those that failed, etc. In particular, when a node needs to access some wire w, that was previously stored, the node knows exactly which LDC codeword contains it, and which index of that codeword it should decode in order to retrieve w.

▶ **Lemma 14** (Bookkeeping)**.** *Any non-crashed node knows, at the start of any round, the following information: (1) the set S of gates whose outputs were stored in the network, and (2) for any $g \in S$ and any output w of g, the LDC codeword that contains w (namely, the node v_j that stored it and the round in which it was stored) and the index of w in the string U_j that v_j stored.*

Proof. Recall that, by definition, since a crashed node does not send any messages starting from the round in which it crashes, all nodes know the set $\mathcal{A}$ of non-crashed nodes at the beginning of every round.

Algorithm 5 Robust Circuit Computation (for node v_j).

Inputs:
A globally known circuit C.
The inputs wires(0) to the circuit C are stored in the network via the Store procedure.
Global parameters Λ, maxRetTime and maxStoreTime (to be set later).

1: **for** Layer $i = 1, \ldots, d$ **do** ▷ The layer-loop
2: $rep \leftarrow 1$, $S \leftarrow \emptyset$
3: **for** $\ell_1 = 1, \ldots, \lceil \log n \rceil$ **do** ▷ The NodeDoubling-loop
4: $G \leftarrow \mathsf{gates}(i) \setminus S$
5: $G_j \leftarrow \mathsf{Allocate}(G, \ell_1)$
6: Let W_j be the set of wires required for computing all gates in G_j.
7: **for** $\ell_2 = 1, \ldots, \lceil \log \Lambda \rceil$ **do** ▷ The AttemptDoubling-loop
8: Execute BulkRetrieve(W_j, ℓ_2). ▷ Idle until maxRetTime rounds pass
9: Remove from W_j all wires that were successfully retrieved in line 8.
10: **if** $W_j = \emptyset$ **then** ▷ Otherwise, idle
11: Let U_j be the output wires of G_j.
12: Locally compute the values of U_j using the retrieved values.
13: Execute Store(U_j). ▷ Idle until maxStoreTime rounds pass
14: **end if**
15: Update S to include gates whose output wires were stored by at least one node.
▷ If too many failures have occurred, repeat layer i
16: Let $f_{i,rep}$ be the number of crashes since the last execution of line 2 or 19.
17: **if** $f_{i,rep} > c_f n/(q \log n)$ **then**
18: $rep \leftarrow rep + 1$
19: **continue from** line 3, re-setting $\ell_1 \leftarrow 1$.
20: **end if**
21: **end for**
22: **end for**
23: **end for**

We prove Items (1) and (2) by induction on the round number. At initialization (the beginning of the first round, $r = 1$), the inputs to the circuit C are assumed to be stored, hence (1) and (2) hold for the input gates gates(0).

Next, we assume the statement holds at the beginning of some NodeDoubling-step, and we show it holds in every round until the end of this NodeDoubling-step. Note that all nodes execute Algorithm 5 in synchrony and, specifically, they all perform BulkRetrieve or Store at the same rounds (other actions do not involve communication as they are purely computational and thus take zero rounds).

If round r is not the final round of a Store procedure, then the set of stored wires is unchanged. Otherwise, each node knows the set S of stored gates at the beginning of the Store, by the induction hypothesis, and thus it also knows the set $G = \mathsf{gates}(i) \setminus S$. Since Allocate is deterministic and depends only on C, ℓ_1, G, and S, then all nodes learn the same output of Allocate(G, ℓ_1). In particular, they all learn the gates G_j that each $v_j \in \mathcal{A}$ is assigned to compute in this NodeDoubling-step.

With this knowledge, all nodes can (locally) generate the good randomness strings that are used by some v_j for each of its BulkRetrieve invocations (Lemma 10). Further, all nodes have the same knowledge about Retrieve calls that failed in previous rounds due to new

crashes. They can thus infer which nodes v_j have successfully retrieved all input wires of G_j (i.e., those for which $W_j = \emptyset$) and satisfy the condition of in Algorithm 5. Only these nodes perform the Store that completes in that round r.

Out of the nodes that perform Store, any node v_j that does not crash before round r, succeeds in storing all the wires in U_j (i.e., all the output wires of G_j).

Therefore, at the start of round $r + 1$, all the nodes in $\mathcal{A}$ learn the set of nodes that performed a successful Store. They also know the set U_j of each v_j that completed a Store, in particular, which wires it contains and their internal order.[2] This implies Item (2). It also follows that all nodes in $\mathcal{A}$ can update their set S of stored gates in a consistent manner (line 15), which implies Item (1). ◀

As a result of this careful bookkeeping, v_j does not need to send any "metadata" information to the nodes it needs to query – they already have all the needed information (in the notations of BulkRetrieve, they know w and $U_{(w)}$, the randomness strings $\mathcal{R}(v_j)$, and the specific round(s) in which v_j queries them (i.e., is expecting a symbol from them).

This concludes the description of the algorithm. The algorithm's analysis, the description of the LDC we construct for the purpose of deterministic local decoding, and the proof of Lemma 10 are deferred to the full version of the paper [8].

References

1 Yagel Ashkenazi, Ran Gelles, and Amir Leshem. Noisy beeping networks. *Information and Computation*, 289:104925, 2022. doi:10.1016/j.ic.2022.104925.

2 Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004.

3 John Augustine, Anisur Rahaman Molla, Gopal Pandurangan, and Yadu Vasudev. Byzantine connectivity testing in the congested clique. In *36th International Symposium on Distributed Computing (DISC)*, volume 246, pages 7:1–7:21, 2022. doi:10.4230/LIPIcs.DISC.2022.7.

4 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 243–252, 2020. doi:10.1145/3382734.3404504.

5 Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2426–2458, 2022. doi:10.1137/1.9781611977073.97.

6 Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 34(6):463–487, 2021. doi:10.1007/s00446-020-00380-5.

7 Keren Censor-Hillel, Orr Fischer, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Quantum distributed algorithms for detection of cliques. In *13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215, pages 35:1–35:25, 2022. doi:10.4230/LIPIcs.ITCS.2022.35.

8 Keren Censor-Hillel, Orr Fischer, Ran Gelles, and Pedro Soto. Two for one, one for all: Deterministic ldc-based robust computation in congested clique, 2025. doi:10.48550/arXiv.2508.08740.

9 Keren Censor-Hillel, Orr Fischer, Tzlil Gonen, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Fast distributed algorithms for girth, cycles and small subgraphs. In *34th International Symposium on Distributed Computing (DISC)*, volume 179, pages 33:1–33:17, 2020. doi:10.4230/LIPIcs.DISC.2020.33.

[2] While U_j is a set of wire-values, the Store procedure expects a bitstring, hence we induce some standard order on the set U_j that maps it into a bitstring, and this ordering is known by all nodes.

10 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Symposium on Principles of Distributed Computing (PODC)*, pages 474–482, 2020. doi:10.1145/3382734.3405742.

11 Keren Censor-Hillel and Einav Huberman. Near-optimal resilient labeling schemes. In *28th International Conference on Principles of Distributed Systems (OPODIS)*, volume 324, pages 35:1–35:22, 2024. doi:10.4230/LIPIcs.OPODIS.2024.35.

12 Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015. doi:10.1145/2767386.2767414.

13 Keren Censor-Hillel and Pedro Soto. Computing in a faulty congested clique. *CoRR*, abs/2505.11430, 2025. doi:10.48550/arXiv.2505.11430.

14 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of (Δ+1) coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 471–480, 2019. doi:10.1145/3293611.3331607.

15 David Cifuentes-Núñez, Pedro Montealegre, and Ivan Rapaport. Recognizing hereditary properties in the presence of byzantine nodes. *CoRR*, abs/2312.07747, 2023. doi:10.48550/arXiv.2312.07747.

16 Sam Coy, Artur Czumaj, Peter Davies, and Gopinath Mishra. Optimal (degree+1)-coloring in congested clique. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261, pages 46:1–46:20, 2023. doi:10.4230/LIPIcs.ICALP.2023.46.

17 Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in congested clique and MPC. *SIAM J. on Computing*, 50(5):1603–1626, 2021. doi:10.1137/20M1366502.

18 Peter Davies. Optimal message-passing with noisy beeps. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 300–309, 2023. doi:10.1145/3583668.3594594.

19 Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *Distributed Computing*, volume 7611, pages 195–209. Springer, 2012. doi:10.1007/978-3-642-33651-5_14.

20 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

21 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

22 Orr Fischer, Rotem Oshman, and Dana Shamir. Explicit space-time tradeoffs for proof labeling schemes in graphs with small separators. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217, pages 21:1–21:22, 2021. doi:10.4230/LIPIcs.OPODIS.2021.21.

23 Orr Fischer and Merav Parter. Distributed CONGEST algorithms against mobile adversaries. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 262–273, 2023. doi:10.1145/3583668.3594578.

24 Orr Fischer and Merav Parter. All-to-all communication with mobile edge adversary: Almost linearly more faults, for free. *CoRR*, abs/2505.05735, 2025. doi:10.48550/arXiv.2505.05735.

25 Pawel Garncarek, Dariusz R. Kowalski, Shay Kutten, and Miguel A. Mosteiro. Beeping deterministic congest algorithms in graphs. *CoRR*, abs/2502.13424, 2025. doi:10.48550/arXiv.2502.13424.

26 Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2016. doi:10.1145/2933057.2933103.

27 Alex B. Grilo, Ami Paz, and Mor Perry. Distributed non-interactive zero-knowledge proofs. *CoRR*, abs/2502.07594, 2025. doi:10.48550/arXiv.2502.07594.

28 James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and MST. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–100, 2015. doi:10.1145/2767386.2767434.

29 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 381–389, 2017. doi:10.1145/3087801.3087811.

30 Tomasz Jurdzinski and Krzysztof Nowicki. MST in $O(1)$ rounds of congested clique. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018. doi:10.1137/1.9781611975031.167.

31 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 2000. doi:10.1145/335305.335315.

32 Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016. doi:10.48550/arXiv.1605.02022.

33 Manish Kumar. Fault-tolerant graph realizations in the congested clique, revisited. In *Distributed Computing and Intelligent Technology*, volume 13776, pages 84–97. Springer, 2023. doi:10.1007/978-3-031-24848-1_6.

34 Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Fault-tolerant graph realizations in the congested clique. In *Algorithmics of Wireless Networks*, volume 13707, pages 108–122, 2022. doi:10.1007/978-3-031-22050-0_8.

35 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 42–50, 2013. doi:10.1145/2484239.2501983.

36 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. on Computing*, 35(1):120–131, 2005. doi:10.1137/S0097539704441848.

37 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

38 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

39 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–115, 2020. doi:10.1137/1.9781611975994.67.

40 Krzysztof Nowicki. A deterministic algorithm for the MST problem in constant rounds of congested clique. In *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1154–1165, 2021. doi:10.1145/3406325.3451136.

41 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 405–414, 2018. doi:10.1145/3210377.3210409.

42 Merav Parter. (delta+1) coloring in the congested clique model. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, pages 160:1–160:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.160.

43 Merav Parter and Hsin-Hao Su. Randomized (Delta+1)-coloring in O(log* Delta) congested clique rounds. In *32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 39:1–39:18, 2018. doi:10.4230/LIPIcs.DISC.2018.39.

44 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

45 Daniel A. Spielman. Highly fault-tolerant parallel computation. In *Proceedings of 37th Conference on Foundations of Computer Science (FOCS)*, pages 154–163, 1996. doi:10.1109/SFCS.1996.548474.

46 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends® in Theoretical Computer Science*, 6(3):139–255, 2012. doi:10.1561/0400000030.

A Chernoff Bounds

▶ **Theorem 15** (Chernoff inequality for independent Bernoulli variables). *Let $X_1, \ldots, X_n$ be mutually independent 0–1 random variables with $\Pr(X_i = 1) = p_i$. Let $X = \sum_{i=1}^{n} X_i$ and set $\mu = E[X]$. The following holds,*

1. *for any $\delta > 0$, $\Pr(X \geq (1+\delta)\mu) \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$*
2. *for $0 < \delta \leq 1$, $\Pr(X \geq (1+\delta)\mu) \leq e^{-\mu\delta^2/3}$*
3. *for $R \geq 6\mu$, $\Pr(X \geq R) \leq 2^{-R}$*

For proof, see Theorem 4.4 in [38].

B Missing Proofs from Section 3

Proof of Lemma 12. Set $r = |G|$. let $g_1, \ldots, g_r$ be the gates in G sorted in a decreasing order of their fan. For any $i \in [r]$, set $d_i = \mathsf{fan}(g_i)$, then,

$$d_r \leq d_{r-1} \leq \ldots \leq d_1 \leq \Delta.$$

First, we analyze the case where $|\mathcal{A}|/2 \leq L \leq |\mathcal{A}|$. We notice that the load of each vertex v is trivially bounded by $\lambda(v) \leq \sum_{i=1}^{r} d_i$ (which is obtained if all gates are allocated to v). On the other hand, by assumption that $|\mathcal{A}|/2 \leq L$ it follows that

$$\lambda(v) \leq \sum_{i=1}^{r} d_i = P \leq 2PL/|\mathcal{A}|,$$

and the claim for the case of $|\mathcal{A}|/2 \leq L \leq |\mathcal{A}|$ follows.

We assume in the remainder of the proof that $L < |\mathcal{A}|/2$. First, we make the very simple observation that at all times during the procedure, it holds that

$$\sum_{v \in V} \lambda(v) \leq \sum_{i=1}^{r} d_i \cdot L = PL. \tag{1}$$

We split the analysis into two phases of the greedy allocation procedure: we say that the procedure is in its first phase while $d_i \geq 2PL/|\mathcal{A}|$ and we say that it is in its second phase while $d_i < 2PL/|\mathcal{A}|$. In each phase, we show that following a gate being allocated to batch of L vertices, all vertices have load at most $\max(4PL/|\mathcal{A}|, \Delta)$.

While $d_i \geq 2PL/|\mathcal{A}|$, we must have at least $|\mathcal{A}|/2$ vertices with no load. Assume otherwise, then we notice that since the gates $g_1, \ldots, g_r$ are sorted in descending order according to $d_1, \ldots, d_r$, then each vertex with an allocated gate has load at least $d_i \geq 2PL/|\mathcal{A}|$. Summing the loads of all vertices, we conclude that the total load is at least $\frac{2PL}{|\mathcal{A}|}(\frac{|\mathcal{A}|}{2} + 1) > PL$, contradicting Equation (1). Hence, after any allocation where $d_i \geq 2PL/|\mathcal{A}|$, we only pick vertices with $\lambda(v) = 0$, and the load of any such vertex v is therefore at most Δ.

Next, we consider the phase where $d_i \leq 2PL/|\mathcal{A}|$. By an averaging argument on Equation (1), at any point of time of the procedure we have at least $|\mathcal{A}|/2$ vertices with load at most $2PL/|\mathcal{A}|$. Since $L < |\mathcal{A}|/2$, the new load of a vertex v that is assigned a new gate is at most

$$\lambda(v) \leq d_i + \frac{2PL}{|\mathcal{A}|} \leq \frac{4PL}{|\mathcal{A}|},$$

and the claim follows. ◀

Content-Oblivious Leader Election in 2-Edge-Connected Networks

Jérémie Chalopin
Aix Marseille Univ, CNRS, LIS, Marseille, France

Yi-Jun Chang
National University of Singapore, Singapore

Lyuting Chen
National University of Singapore, Singapore

Giuseppe A. Di Luna
DIAG, Sapienza University of Rome, Italy

Haoran Zhou
National University of Singapore, Singapore

Abstract

Censor-Hillel, Cohen, Gelles, and Sela (PODC 2022 & Distributed Computing 2023) studied *fully-defective* asynchronous networks, where communication channels may arbitrarily corrupt messages. The model is equivalent to *content-oblivious* computation, where nodes communicate solely via pulses. They showed that if the network is 2-edge-connected, then any algorithm for a noiseless setting can be simulated in the fully-defective setting; otherwise, no non-trivial computation is possible in the fully-defective setting. However, their simulation requires a predesignated leader, which they conjectured to be necessary for any non-trivial content-oblivious task.

Recently, Frei, Gelles, Ghazy, and Nolin (DISC 2024) refuted this conjecture for the special case of oriented ring topology. They designed two asynchronous content-oblivious leader election algorithms with message complexity $O(n \cdot \mathsf{ID}_{\max})$, where n is the number of nodes and $\mathsf{ID}_{\max}$ is the maximum ID. The first algorithm stabilizes in unoriented rings without termination detection. The second algorithm quiescently terminates in oriented rings, thus enabling the execution of the simulation algorithm after leader election. In this work, we present two results:

General 2-edge-connected topologies: First, we show an asynchronous content-oblivious leader election algorithm that quiescently terminates in any 2-edge-connected network with message complexity $O(m \cdot N \cdot \mathsf{ID}_{\min})$, where m is the number of edges, N is a known upper bound on the number of nodes, and $\mathsf{ID}_{\min}$ is the smallest ID. Combined with the above simulation, this result shows that whenever a size bound N is known, any noiseless algorithm can be simulated in the fully-defective model without a preselected leader, fully refuting the conjecture.

Unoriented rings: We then show that the knowledge of N can be dropped in unoriented ring topologies by presenting a quiescently terminating election algorithm with message complexity $O(n \cdot \mathsf{ID}_{\max})$ that matches the previous bound. Consequently, this result constitutes a *strict* improvement over the previous state of the art and shows that, on rings, fully-defective and noiseless communication are computationally equivalent, with no additional assumptions.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Asynchronous model, fault tolerance, quiescent termination

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.21

Related Version *Full Version*: https://arxiv.org/abs/2507.08348 [9]

Funding This work has been partially supported by ANR project DUCAT (ANR-20-CE48-0006), by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (24-1323-A0001), and by the Italian MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU through projects SERICS (PE00000014).



© Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Fault tolerance is a cornerstone of distributed computing, enabling systems to remain operational despite various failures, such as node crashes or channel noise [5, 17, 32, 38]. One important category of faults, known as *alteration errors*, changes the content of messages but does not create new messages or eliminate existing ones. Traditional methods for handling such errors often rely on adding redundancy through coding techniques and assume bounds on the frequency or severity of faults. However, in many practical settings, such assumptions may not hold, limiting the effectiveness of these approaches.

Fully-defective networks. Censor-Hillel, Cohen, Gelles, and Sela [7] introduced *fully-defective* asynchronous networks, in which all links are subject to severe alteration errors: The content of every message can be arbitrarily modified, although the adversary cannot insert new messages or remove existing messages. Since messages can no longer carry meaningful information, the model is equivalent to *content-oblivious* computation, where communication is reduced to sending and receiving *pulses*. Algorithms in this setting operate entirely based on the patterns of pulse arrivals.

An impossibility result. A natural approach to designing a content-oblivious algorithm is to use unary encoding, representing messages as sequences of pulses. However, in the *asynchronous* setting, where no upper bound on message delivery time is known, there is no reliable way to detect the end of a sequence. Censor-Hillel, Cohen, Gelles, and Sela [7] established a strong *impossibility* result: Let $f(x, y)$ be any non-constant function, then any two-party deterministic asynchronous content-oblivious communication protocol, where one party holds x and the other holds y, must fail to compute f correctly.

The impossibility result implies that, in a certain sense, no non-trivial computation is possible in any network that is *not* 2-edge-connected – that is, any network containing an edge e whose removal disconnects the graph into two components. By assigning control of each component to a different party, the problem reduces to the two-party setting to which the impossibility result applies.

Algorithm simulation over 2-edge-connected networks. Interestingly, Censor-Hillel, Cohen, Gelles, and Sela [7] showed that in 2-edge-connected networks, any algorithm designed for the noiseless setting can be simulated in a fully defective network. We briefly sketch the main ideas underlying their approach below.

For the special case of an oriented ring, the challenge of termination detection can be addressed. Suppose a node wishes to broadcast a message. It can send a sequence of pulses that represents the unary encoding of the message in one direction and use the opposite direction to signal termination. To allow all nodes the opportunity to communicate, a token can be circulated in the ring, enabling each node to speak in turn.

To extend this approach to general 2-edge-connected networks, they leverage a result by Robbins [39], which states that any 2-edge-connected undirected graph can be oriented to form a strongly connected directed graph. This guarantees the existence of a *Robbins cycle*, which is an oriented ring that traverses all nodes, possibly revisiting some nodes multiple times. They designed a content-oblivious algorithm that uses a depth-first search (DFS) to construct such a cycle.

Preselected leader assumption. A key limitation of their result is that it inherently relies on a *preselected leader*. For instance, a leader is needed to generate a unique token in a ring or to select the root for the DFS. Censor-Hillel, Cohen, Gelles, and Sela [7] conjectured that the preselected leader assumption is essential for any non-trivial content-oblivious computation, leaving the following question open.

▶ **Question 1.** *In 2-edge-connected networks, is it possible to simulate any algorithm designed for a noiseless setting in the fully defective model without relying on a preselected leader?*

Content-oblivious leader election on rings. To answer the above question affirmatively, it suffices to design a content-oblivious leader election algorithm that possesses certain desirable properties, enabling it to be composed with other algorithms. Recently, Frei, Gelles, Ghazy, and Nolin [24] answered this question for the special case of *oriented rings*. They designed two asynchronous content-oblivious leader election algorithms, both with message complexity $O(n \cdot \mathsf{ID}_{\max})$, where n is the number of nodes and $\mathsf{ID}_{\max}$ is the maximum identifier. The first algorithm *stabilizes* in unoriented rings, without termination detection. The second algorithm *quiescently* terminates in oriented rings, ensuring that no further pulses are sent to a node v after the algorithm on v declares termination.

The quiescent termination property enables the execution of the general simulation algorithm discussed earlier, following their leader election algorithm, as each node can correctly identify which pulses belong to which algorithm. As a result, this finding refutes the conjecture of Censor-Hillel, Cohen, Gelles, and Sela [7] for the special case of oriented ring topologies. Given the work by Frei, Gelles, Ghazy, and Nolin [24], the next question to address is whether content-oblivious leader election is feasible for general 2-edge-connected networks.

▶ **Question 2.** *In 2-edge-connected networks, is it possible to design a quiescently terminating content-oblivious leader election algorithm?*

Existing approaches for content-oblivious leader election heavily rely on the *oriented* ring structure, making use of its two orientations to handle two types of messages. We briefly outline the key ideas behind the algorithms of Frei, Gelles, Ghazy, and Nolin [24] as follows.

A central building block of their algorithms is a stabilizing leader election algorithm on oriented rings, which operates as follows. Each node generates a token, and all tokens travel in the same direction. Each node v maintains a counter to track the number of tokens it has passed. Once the counter reaches $\mathsf{ID}(v)$, node v destroys one token and temporarily elects itself as leader. Any incoming token will cause it to relinquish leadership. All counters eventually reach $\mathsf{ID}_{\max}$ and stabilize. The node r with $\mathsf{ID}(r) = \mathsf{ID}_{\max}$ becomes the unique leader. This algorithm is not terminating, as nodes cannot determine whether additional tokens will arrive.

To achieve a quiescently terminating algorithm on oriented rings, they run the procedure in both directions sequentially, using the moment when the two counter values match as the termination condition. A node v begins the second run when its counter value from the first run reaches $\mathsf{ID}(v)$.

We stress that orientation is a key assumption of the above algorithm. In fact, they conjectured the impossibility of terminating leader election in unoriented rings [24].

To achieve a stabilizing algorithm on unoriented rings, they run the procedure in both directions concurrently. The correctness follows from the observation that the same node is selected as the leader in both executions.

1.1 Contributions

We answer Question 1 affirmatively under the assumption that an upper bound $N \geq n$ on the number of nodes $n = |V|$ is known, by presenting a content-oblivious leader election algorithm that quiescently terminates on any 2-edge-connected network.

▶ **Theorem 1.** *There is a quiescently terminating leader election algorithm with message complexity $O(m \cdot N \cdot \mathsf{ID}_{\min})$ in any 2-edge-connected network $G = (V, E)$, where $m = |E|$ is the number of edges, $N \geq n$ is a known upper bound on the number of nodes $n = |V|$, and $\mathsf{ID}_{\min}$ is the smallest ID. Moreover, the leader is the last node to terminate.*

Combined with the simulation result of Censor-Hillel, Cohen, Gelles, and Sela [7], our finding implies that in any 2-edge-connected network, any algorithm designed for the noiseless setting can be simulated in the fully-defective setting, without assuming a preselected leader. More precisely, as established in the prior work [24], a sufficient condition for achieving this objective consists of (1) quiescent termination and (2) the requirement that the leader is the final node to terminate. This is because the algorithmic simulation in [7] is initiated by the leader. Hence we answer Question 2 affirmatively, again under the assumption that an upper bound $N \geq n$ is known. This refutes the original conjecture posed by [7].

For unoriented ring topologies, the knowledge of N is not required.

▶ **Theorem 2.** *There exists a quiescently terminating leader election algorithm with message complexity $O(n \cdot \mathsf{ID}_{\max})$ in any unoriented ring, where $\mathsf{ID}_{\max}$ is the largest identifier. Moreover, the leader is the last node to terminate.*

This result refutes the conjecture of Frei, Gelles, Ghazy, and Nolin [24], demonstrating that orientation is not necessary for quiescently terminating leader election in rings.

Table 1 New and old results on content-oblivious leader election.

Topology	Guarantee	Messages	Condition	Reference
Unoriented rings	Stabilizing	$O\left(n \cdot \mathsf{ID}_{\max}\right)$	$\times$	[24]
Oriented rings	Quiescently terminating	$O\left(n \cdot \mathsf{ID}_{\max}\right)$	$\times$	[24]
Oriented rings	Stabilizing	$\Omega\left(n \cdot \log \frac{\mathsf{ID}_{\max}}{n}\right)$	$\times$	[24]
2-edge-connected	Quiescently terminating	$O\left(m \cdot N \cdot \mathsf{ID}_{\min}\right)$	Known N	Theorem 1
Unoriented rings	Quiescently terminating	$O\left(n \cdot \mathsf{ID}_{\max}\right)$	$\times$	Theorem 2

See Table 1 for a comparison of our results and the results from prior work [24].

Our algorithm for general 2-edge-connected topologies has a higher message complexity of $O(m \cdot N \cdot \mathsf{ID}_{\min})$ compared to the previous $O(n \cdot \mathsf{ID}_{\max})$ bound in terms of network size, but it reduces the dependency on identifiers from $\mathsf{ID}_{\max}$ to $\mathsf{ID}_{\min}$.

At first glance, the upper bound of $O(m \cdot N \cdot \mathsf{ID}_{\min})$ may seem to contradict the lower bound of $\Omega\left(n \cdot \log \frac{\mathsf{ID}_{\max}}{n}\right)$ established in previous work [24]. However, the two results are consistent, as the lower bound does not make any assumptions about the value of $\mathsf{ID}_{\min}$. Precisely, their bound is of the form $\Omega\left(n \cdot \log \frac{k}{n}\right)$, where k is the number of distinct, *assignable* identifiers in the network. In other words, k is the size of the ID space.

While our algorithm for Theorem 1 solves leader election in any 2-edge-connected topology, it does require a mild assumption: All nodes must *a priori* agree on a known upper bound N on the number of nodes $n = |V|$ in the network, making it *non-uniform*. This assumption is widely adopted in many existing leader election and distributed graph algorithms, as upper bounds on network size are often available or can be estimated. In contrast, the terminating algorithm from prior work [24] is *uniform*, but only works on oriented rings.

Our leader election algorithm on rings requires neither orientation nor knowledge of N, while matching the message complexity of the algorithms of Frei, Gelles, Ghazy, and Nolin [24]. It therefore constitutes a *strict* improvement over the previous state of the art.

2 Preliminaries

We present the model and problem considered in the paper and overview some basic terminology.

2.1 Content-Oblivious Model

The communication network is a graph $G = (V, E)$, where each node $v \in V$ is a computing device and each edge $e \in E$ is a bidirectional communication link. We write $n = |V|$ and $m = |E|$. Each node v has a unique identifier $\mathsf{ID}(v)$, and we define $\mathsf{ID}_{\max} = \max_{v \in V} \mathsf{ID}(v)$ and $\mathsf{ID}_{\min} = \min_{v \in V} \mathsf{ID}(v)$. Throughout the paper, we assume that G is 2-edge-connected: removing one edge does not disconnect the graph.

General 2-edge-connected topologies. For our result on general 2-edge-connected topologies, we assume no prior knowledge of the network topology, except that each node is given an upper bound $N \geq n$ on the total number of nodes. That is, the algorithm is non-uniform.

The degree of a node v, $\deg(v)$, is the number of edges incident to v. A node does not know the identifiers of its neighbors, but can distinguish its incident edges using *port numbering*, a bijection between the set of incident edges and the set $\{1, 2, \ldots, \deg(v)\}$. Therefore, we use the term port to refer to the local endpoint of an edge, uniquely identified by its port number, which is used to send and receive messages to and from a specific neighbour.

Unoriented rings. For our result on rings, we assume that the topology is an unoriented ring. In this case, each node v_i is connected to nodes v_{i-1} and v_{i+1} (indices taken modulo n) by port 0 and port 1. These local labels do not induce a consistent orientation of the ring; that is, the global assignment of labels 0 and 1 to ports is arbitrarily set by an adversary. A correct algorithm must work for any possible assignment of port labels.

Content-Oblivious communication. In the content-oblivious setting, nodes communicate by sending *content-less pulses*. We consider the *asynchronous* model, where node behavior is *event-driven*: A node can act only upon initialization or upon receiving a pulse. Based on the pattern of previously received pulses, each node decides its actions, which may include sending any number of pulses through any subset of its ports. We emphasize that when a node receives a pulse, it knows the port through which the pulse arrived. We restrict our attention to the *deterministic* setting, in which nodes do not use randomness in their decision-making.

Time. There is no upper bound on how long a pulse may take to traverse an edge, but every pulse is guaranteed to be delivered after a finite delay. An equivalent way to model this behavior is by assuming the presence of a scheduler. At each time step, as long as there are pulses in transit, the scheduler arbitrarily selects one and delivers it.

We assume that upon receiving a pulse, all actions triggered by that pulse are performed *instantaneously*, so that the system can be analyzed in *discrete* time steps. We focus only on specific, well-defined moments in the execution of the algorithm – namely, the moments just

before the scheduler selects a pulse to deliver. At these times, all the actions resulting from earlier pulse deliveries have been completed, and the system is in a stable state, awaiting the next step. We do not analyze the system during any intermediate period in which a node is still performing the actions triggered by a pulse arrival.

Counting the number of pulses. We write $[s] = \{1, 2, \ldots, s\}$. In the algorithm description and analysis for Theorem 1, for each node $v \in V$ and each port number $i \in [\deg(v)]$, we use the following notation to track the number of pulses sent and received so far:

- $\sigma_i(v)$ denotes the number of pulses sent on port i of node v.
- $\rho_i(v)$ denotes the number of pulses received on port i of node v.

2.2 Leader Election

We now define the *leader election* problem.

▶ **Definition 3** (Leader election). *The task of leader election requires that each node outputs exactly one of* Leader *and* Non-Leader*, with the additional requirement that exactly one node outputs* Leader*.*

More precisely, each node maintains a binary internal variable isLeader to represent its current output, taking the value Leader or Non-Leader. The value of isLeader may change arbitrarily many times during the execution of the algorithm, but it must eventually stabilize to a final value that remains unchanged thereafter. The final outputs of all nodes must collectively satisfy the conditions of Definition 3.

Due to the asynchronous nature of our model, an algorithm can *finish* in various ways. We hence classify algorithms into three types by the increasing strength of termination guarantees.

▶ **Definition 4** (Stabilization). *A node stabilizes at time t if its output does not change after time t. An algorithm is stabilizing if it guarantees that all nodes eventually stabilize.*

▶ **Definition 5** (Termination). *A node terminates at time t if it explicitly declares termination at time t. After declaring termination, the node ignores all incoming pulses and can no longer change its output. An algorithm is terminating if it guarantees that all nodes eventually terminate.*

▶ **Definition 6** (Quiescent termination). *A node quiescently terminates at time t if it terminates at time t and no further pulses arrive at the node thereafter. An algorithm is quiescently terminating if it guarantees that all nodes eventually quiescently terminate.*

Ideally, we aim to design leader election algorithms that are quiescently terminating, as this property allows a second algorithm to be executed safely after leader election. In contrast, if the algorithm guarantees only stabilization or termination, nodes may be unable to distinguish whether an incoming pulse belongs to the leader election process or the subsequent algorithm. This ambiguity can affect the correctness of both algorithms. For this reason, prior work [24] establishes a general algorithm simulation result only for oriented rings, but not for unoriented rings.

2.3 DFS and Strongly Connected Orientation

We describe a DFS tree T_G of the graph $G = (V, E)$ that is *uniquely determined* by the port numbering and node identifiers. Based on this DFS tree, we describe an edge orientation of G that transforms it into a strongly connected directed graph $\vec{G}$. Both the DFS tree and the orientation play a critical role in the analysis of our algorithm for Theorem 1.

DFS. Throughout this paper, we write r to denote the node with the smallest identifier. Consider a DFS traversal rooted at r such that when a node selects the next edge to explore among its unexplored incident edges, it chooses the one with the *smallest* port number. Let $T_G = (V, E_T)$ be the resulting DFS tree. The edges in E_T are called *tree edges*, and the remaining edges $E_B = E \setminus E_T$ are called *back edges*. The following observation is folklore.

▶ **Observation 7.** *For any edge $\{u, v\} \in E_B$, either u is an ancestor of v, or v is an ancestor of u.*

Proof. When a node u explores an incident edge $e = \{u, v\}$ during the DFS, there are two possibilities: Either v is an ancestor of u, or v has not been visited yet. In the first case, $\{u, v\} \in E_B$. In the second case, v is discovered for the first time, so $\{u, v\} \in E_T$ is added as a tree edge in E_T. ◀

▶ **Definition 8** (Strongly connected orientation)**.** *The directed graph $\vec{G} = (V, \vec{E}_T \cup \vec{E}_B)$ is defined by*

$$\vec{E}_T = \{(u, v) \mid \{u, v\} \in E_T \text{ and } u \text{ is the parent of } v \text{ in } T_G\},$$
$$\vec{E}_B = \{(u, v) \mid \{u, v\} \in E_B \text{ and } v \text{ is an ancestor of } u \text{ in } T_G\}.$$

A directed graph is *strongly connected* if for any two nodes u and v in the graph, there is a directed path from u to v. We have the following observation. Due to page limit, the proof of Observation 9 is deferred to the full version [9] of the paper.

▶ **Observation 9.** *If G is 2-edge-connected, then $\vec{G}$ is strongly connected.*

Shortest paths. Throughout the paper, for any two nodes $u \in V$ and $v \in V$, we write $P_{u,v}$ to denote any shortest path from u to v in $\vec{G}$, whose existence is guaranteed by Observation 9. We allow $u = v$, in which case $P_{u,v}$ is a path of zero length consisting of a single node $u = v$.

3 Non-Uniform Leader Election for 2-Edge-Connected Topologies

In this section, we provide a high-level proof sketch for Theorem 1; the pseudocode is deferred to Section B, and the complete proof is deferred to the full version [9] of the paper.

▶ **Theorem 1.** *There is a quiescently terminating leader election algorithm with message complexity $O(m \cdot N \cdot \mathsf{ID}_{\min})$ in any 2-edge-connected network $G = (V, E)$, where $m = |E|$ is the number of edges, $N \geq n$ is a known upper bound on the number of nodes $n = |V|$, and $\mathsf{ID}_{\min}$ is the smallest ID. Moreover, the leader is the last node to terminate.*

Our algorithm aims to elect the node r with the smallest identifier as the leader. It operates in two phases. In the first phase, each node maintains a counter that is incremented in an almost synchronized manner using the *α-synchronizer* of Awerbuch [4]. A node v declares itself the leader once its counter reaches $\mathsf{ID}(v)$. This first phase alone yields a simple *stabilizing* leader election. To ensure *quiescent termination*, the second phase is executed, in which the elected leader announces its leadership through a DFS traversal.

3.1 Synchronized Counting

Counting. To achieve an almost synchronized counting, we employ the α-synchronizer [4], as follows. Each node maintains a counter variable $\mathsf{Count}(v)$, which is initialized to 0 at the start of the algorithm. A node increments its counter by one after receiving a new pulse from each of its neighbors. Formally, the counter value of a node v is given by

$$\mathsf{Count}(v) = \min_{j \in [\deg(v)]} \rho_j(v).$$

Each node broadcasts a pulse to all its neighbors upon initialization and immediately after each counter increment. This process is *non-blocking*: In the absence of an exit condition, nodes will continue counting together indefinitely. Furthermore, no node can advance significantly faster than its neighbors: For any pair of adjacent nodes, their counter values differ by at most one. A faster-counting node must wait for a pulse from a slower-counting neighbor before it can increment its counter again.

Stabilizing leader election. A node v elects itself as the leader once $\mathsf{Count}(v) = \mathsf{ID}(v)$, after which node v *freezes* its counter and does not increment it further. This action caps the counter value of any neighbor of v at most $\mathsf{ID}(v) + 1$. More generally, in a graph of $n \leq N$ nodes, the counter value of any node is upper bounded by $\mathsf{ID}(v) + N - 1$, due to the diameter upper bounded by $N - 1$.

To ensure that the algorithm elects the node r with the smallest identifier as the *unique* leader, it suffices for any two identifiers to differ by at least N. This can be achieved by first multiplying each identifier by N. This is where the *a priori* knowledge of N comes into play.

At this stage, we have a simple stabilizing leader election algorithm. However, it remains insufficient for our purposes: We also want all non-leader nodes to recognize their non-leader status and terminate in a quiescent manner.

3.2 DFS Notification

We proceed to describe the second phase of the algorithm. The leader must find a way to notify all other nodes to settle as non-leaders. This seems challenging because the leader can only send additional pulses, which appear indistinguishable from those in the first phase. As we will see, assuming the network is 2-edge-connected, there is a simple and effective approach: Just send many pulses to overwhelm the listener.

Notifying a neighbor. Let u be a neighbor of the leader r, and let the edge $e = \{u, r\}$ correspond to port i at r and port j at u. The leader r notifies u of its leadership as follows:

1. Wait until $\rho_i(r) \geq \mathsf{ID}(r) + 1$.
2. Send pulses to u until $\sigma_i(r) = \mathsf{ID}(r) + N + 2$.

Since the graph G is 2-edge-connected, there exists a path P from u to r that avoids e, implying

$$\mathsf{Count}(u) \leq \mathsf{Count}(r) + \mathsf{length}(P) \leq \mathsf{ID}(r) + N - 1.$$

Eventually, u receives $\mathsf{ID}(r) + N + 2$ pulses from r, so there must be a moment where

$$\rho_j(u) \geq \mathsf{Count}(u) + 3,$$

triggering anomaly detection. Under normal conditions in the first phase, u would expect

$$\rho_j(u) \leq \sigma_i(r) = \mathsf{Count}(r) + 1 \leq \mathsf{Count}(u) + 2,$$

so this deviation signals a violation.

Additionally, the waiting condition $\rho_i(r) \geq \mathsf{ID}(r) + 1$ ensures that $\mathsf{Count}(u) \geq \mathsf{ID}(r)$. Thus, u not only recognizes that it is not the leader but can also deduce the exact value of $\mathsf{ID}(r)$ using

$$\mathsf{ID}(r) = N \cdot \left\lfloor \frac{\mathsf{Count}(u)}{N} \right\rfloor,$$

since all node identifiers are integer multiples of N.

DFS. Intuitively, the leader r is able to notify a neighbor u because the counter value of node u is constrained by a *chain* P, whose other end is *anchored* at the leader r, whose counter has already been frozen. To extend this notification process to the entire network, we perform a DFS, using the same notification procedure to explore each new edge. When a node finishes exploring all its remaining incident edges, it can notify its parent by sending pulses until a total of $\mathsf{ID}(r)+N+2$ pulses have been sent through this port since the start of the algorithm. When a node u explores a back edge $\{u,v\}$, the node v replies using the same method. When a node decides which new incident edge to explore, it prioritizes the edge with the smallest port number, so the resulting DFS tree is T_G, as defined in Section 2.3.

Quiescent termination. To see that the algorithm achieves quiescent termination, observe that by the end of the execution, for each edge e, the number of pulses sent in both directions is exactly $\mathsf{ID}(r)+N+2$. Therefore, once a node observes that it has both sent and received exactly $\mathsf{ID}(r)+N+2$ pulses along each of its ports and has completed all local computations, it can safely terminate with a quiescent guarantee.

Correctness. A similar chain-and-anchor argument can be applied to analyze the correctness of the DFS-based notification algorithm. We prove by induction that for each node u that has been visited by DFS, its frozen counter value satisfies

$$\mathsf{Count}(u) \le \mathsf{ID}(r) + \mathsf{length}(P_{u,r}).$$

Recall from Section 2.3 that $P_{a,b}$ is a shortest path from a to b in the directed version $\vec{G}$ of G defined in Definition 8.

Suppose during the DFS, a node x is about to explore an incident edge $e=\{x,y\}$ using the notification procedure, and that y has not yet been visited. Let w be the first node on the path $P_{y,r}$ that is either x or an ancestor of x in the DFS tree T_G, then we have

$$\mathsf{length}(P_{y,r}) = \mathsf{length}(P_{y,w}) + \mathsf{length}(P_{w,r}).$$

We now apply the chain-and-anchor argument using the chain $P_{y,w}$ and anchor w. This choice of chain is valid because the path $P_{y,w}$ cannot include the edge e, as e is oriented from x to y in $\vec{G}$. This is precisely why the analysis is carried out in $\vec{G}$ rather than in G.

By the induction hypothesis, the frozen counter value at the anchor satisfies $\mathsf{Count}(w) \le \mathsf{ID}(r) + \mathsf{length}(P_{w,r})$. It follows that

$$\begin{aligned}\mathsf{Count}(y) &\le \mathsf{Count}(w) + \mathsf{length}(P_{y,w})\\ &\le \mathsf{ID}(r) + \mathsf{length}(P_{w,r}) + \mathsf{length}(P_{y,w})\\ &= \mathsf{ID}(r) + \mathsf{length}(P_{y,r})\\ &\le \mathsf{ID}(r) + N - 1.\end{aligned}$$

Therefore, the chain-and-anchor argument works, allowing y to be successfully notified. Moreover, the frozen counter value of y also satisfies the induction hypothesis.

In addition, we must ensure that the DFS procedure proceeds correctly and no event is triggered unexpectedly. Once again, the chain-and-anchor argument implies that the condition

$$\mathsf{Count}(u) \le \mathsf{ID}(r) + \mathsf{length}(P_{u,r}) \le \mathsf{ID}(r) + N - 1 < \mathsf{ID}(u)$$

holds for all nodes $u \in V \setminus \{r\}$ at all times. This ensures that no node other than r can be mistakenly elected as a leader. It also guarantees that pulses sent by nodes still in the first

phase cannot accidentally trigger a notification in the second phase. Specifically, if u is still in the first phase, then for every port $j \in [\deg(u)]$, we have

$$\sigma_j(u) = \mathsf{Count}(u) + 1 \leq \mathsf{ID}(r) + N,$$

which remains below the threshold of $\mathsf{ID}(r) + N + 2$ pulses required to confirm receipt of a notification.

Remark. One might wonder why the notification must follow a DFS instead of being performed in parallel. A key reason is that doing so can break the chain-and-anchor argument by eliminating anchors. For instance, in a ring topology, if the node r with the smallest identifier simultaneously sends multiple pulses to *both* neighbors, all of the pulses might be interpreted as part of the first phase, with no detectable anomaly, regardless of how many pulses are sent.

Both our leader election algorithm and the simulation algorithm of [7] utilize DFS, but for different purposes. They perform a DFS from a pre-selected leader to find a cycle, while we use DFS to disseminate the identity of the elected leader to all nodes and to achieve quiescent termination. Coincidentally, both approaches leverage the strongly connected orientation of 2-edge-connected graphs. In our case, the orientation $\vec{G}$ is used solely in the analysis to facilitate the chain-and-anchor argument. In their case, it guarantees the existence of a Robbins cycle, which is critical to their algorithm.

4 Uniform Leader Election for Unoriented Rings

In this section, we prove our main theorem on rings.

▶ **Theorem 2.** *There exists a quiescently terminating leader election algorithm with message complexity $O(n \cdot \mathsf{ID}_{\max})$ in any unoriented ring, where $\mathsf{ID}_{\max}$ is the largest identifier. Moreover, the leader is the last node to terminate.*

The proof is given by our uniform algorithm, whose pseudocode is given in Algorithm 1.

We introduce a new action, **RcvPulse$_i$**: Wait until a pulse is received from port i, where $i \in \{0, 1, q\}$. When $i = q$, the procedure waits for the first pulse on any port and stores the corresponding port number in the variable q. We also use **SendPulses$_i(k)$**: send k pulses through port i, where $i \in \{0, 1\}$.

We define the local variable **Diff(v)**, which stores the difference between the number of pulses node v received on port 1 and the number of pulses received on port 0.

Algorithm description. Our algorithm is decomposed into different phases that we describe below.

- The *initialization*. First, each node doubles its identifier ID (Line 1). The purpose of this action is to ensure that no two nodes have consecutive identifiers. This allows us to distinguish the different phases of the algorithm during the execution.
- The *competing* phase (Lines 2 to 5 in blue). Every node starts by exchanging pulses with both its neighbors in a pseudo-synchronous way (sending them pulses and waiting for their responses). The number of times a node repeats this step is equal to the value of its identifier ID.

- The *solitude-checking* phase (Lines 7 to 9 in orange). Once a node has finished the competing phase, it wants to know if it was the last one competing. To do so, it sends two pulses on port 0, receives one on port 1 and waits for another pulse. If this last pulse arrives on port 1, it means that its own pulse has made a complete traversal of the cycle and that all other nodes stopped competing. Otherwise (i.e., if it gets a pulse on port 0 before getting two pulses on port 1), it means that there is still another node executing the competing phase.
- The *global termination* phase (Lines 11 to 12 in green). If a node detects that all other nodes stopped competing (i.e., if $q = 1$ on Line 10), then it knows it will be the leader. Before terminating, it sends a pulse on port 0 to inform the other nodes that the execution is over. This termination pulse will traverse the entire ring and come back through port 1. At this point, every other node will eventually be in state Non-Leader and there will be no pulses in transit. Then the node can enter the state Leader and terminate.

Algorithm 1 Uniform leader election on unoriented rings for node v.

```
1  ID(v) ← 2 * ID(v)
2  for i ← 1 to ID(v) do
3  |  SendAll(1)                                          // send 1 pulse on both ports
4  |  RcvPulse_0                                         // wait for a pulse from port 0
5  |  RcvPulse_1                                         // wait for a pulse from port 1
6  end
   /* Check if I am alone, i.e., if I have the largest identifier                    */
7  SendPulses_0(2)                                       // send 2 pulses on port 0
8  RcvPulse_1                                            // wait for a pulse from port 1
9  RcvPulse_q                  // wait for first pulse from any port; store port in q
10 if q = 1 then
   |  /* I am the last one competing                                                 */
11 |  SendPulses_0(1)                  // send termination pulse on port 0 - I'm leader
12 |  RcvPulse_1                      // wait for termination pulse to return from port 1
13 |  return Leader
14 else
   |  /* There is someone with a larger identifier                                   */
15 |  SendPulses_1(2)                          // send 2 pulses on port 1 - balance line 7
16 |  RcvPulse_0                                          // wait for a pulse from port 0
17 |  RcvPulse_1                                          // wait for a pulse from port 1
18 |  Diff ← 0
   |  /* Diff = pulses from port 1 minus pulses from port 0                          */
19 |  repeat
20 |  |  RcvPulse_q                    // wait for pulse from any port; store port in q
21 |  |  Diff ← Diff + 2 * q − 1                          // update difference counter
22 |  |  SendPulses_{1−q}(1)                          // forward pulse to opposite port
23 |  until |Diff| ≥ 3
24 |  return Non-Leader
25 end
```

- The *rebalancing* phase (Lines 15 to 17 in violet). If the node is not alone (i.e., if $q = 0$ on Line 10), we want the node to relay pulses between its competing neighbors so that these neighbors act as if there was no node between them. To do so, we have to ensure that it sent the same number of pulses in both directions. So the node sends two pulses on port 1 (to balance the pulses sent on port 0 on Line 7). At this point, on each port, the node has sent one more pulse than it has received. So it waits for one pulse on each port before entering the relaying phase. For its neighbors, the node behaves as if it has executed two more iterations of the competing phase: it has sent two pulses on each port and has received two pulses on each port. Since no two nodes have consecutive identifiers (recall that each node doubled its ID at Line 1), no other node can terminate its competition phase before receiving both these pulses.

- The *relaying* phase (Lines 18 to 23 in red). In this phase, the node just relays the pulses it gets on port 0 to port 1 and vice-versa. It also keeps a counter (Diff) to store the difference between the numbers of pulses received on port 0 and on port 1. When this counter reaches 3 or -3, then it means that one node has successfully passed the solitude test and sent a termination pulse. The node can then enter the state Non-Leader and terminate.

4.1 Correctness

We say that a node is *active* if it is executing an instruction between Lines 2 and 17. A node is a *relaying node* if is executing an instruction between Lines 18 and 23.

Consider two rings: R of size n and R' of size $n-1$, where R' is obtained by removing a node v from R and connecting its neighbours w and u. We say that an execution on R is indistinguishable from an execution on R' if there exists an execution on R' in which w and u return from waiting on the RcvPulse actions at exactly the same times as they do in R.

In such executions, w and u do not perceive any difference between the execution of the algorithm on R and on R'. This, in turn, implies that all other nodes also observe an execution, when viewed in terms of RcvPulse events, that could occur in either R or R'.

▶ **Proposition 10.** *Consider a ring of size $n \geq 2$ with consecutive nodes v_1, v_2 and v_3 (with $v_1 = v_3$ if $n = 2$) such that v_2 is the node with the smallest identifier on the entire ring. As long as v_2 does not stop, the execution of the algorithm on this ring is indistinguishable for any node on the ring from a reduced ring of size $n-1$, where v_2 has been removed and v_1 is directly connected with v_3.*

Without loss of generality, let us assume that at v_2, the port number to v_1 is 0 and the port number to v_3 is 1. Let ID_1, ID_2, and ID_3 be the respective identifiers of nodes v_1, v_2, v_3. To prove Proposition 10, we first establish several observations and lemmas.

▶ **Observation 11.** *When a node ends the ith iteration of the loop of the competing phase, it has sent i pulses to each of its neighbors and has received i pulses from each of them.*

▶ **Lemma 12.** *Consider two neighboring nodes, v_a and v_b, executing the competing phase. Given their respective indices, i_a and i_b, in the loop at Lines 2–5, we have $|i_a - i_b| \leq 1$. Moreover, if $i_b \leq i_a$ and v_b has not executed the sends of its iteration i_b then $i_b = i_a$.*

Proof. Suppose w.l.o.g. that node v_a exchanges pulses with v_b on port 1, while node v_b exchanges pulses with v_a on port 0.

If node v_a executed x times Line 5 (i.e., $x + 1 \geq i_a \geq x$), then node v_b executed $y \geq x$ times Line 3 (i.e., $i_b \geq y$). From this $i_b \geq i_a - 1$. By a symmetric argument we have $i_a \geq i_b - 1$ and this shows the first part of the lemma. Assume now that v_b has not executed its send, this means that $y = (i_b - 1)$ and thus $x \leq y \leq i_b - 1$ therefore since $i_a \leq x + 1 \leq i_b$. For the hypothesis we have $i_a = i_b$. ◀

▶ **Lemma 13.** *Among nodes v_1, v_2 and v_3, node v_2 is the first to exit the competing phase. Moreover, v_1 (resp. v_3) cannot exit its competing phase before it has received two pulses sent by v_2 after v_2 has exited the competing phase (sent at Line 7 for v_1 and at 15 for v_3).*

Proof. By Lemma 12 and Observation 11, we have that when v_2 exits the loop, it has sent ID_2 pulses to both its neighbors and thus v_1 and v_3 are in iteration ID_2 or $\mathsf{ID}_2 + 1$ of the loop. Since their identifiers are at least $\mathsf{ID}_2 + 2$, they are still in their competing phase. Note that, by Lemma 12 and Observation 11, they cannot complete iteration $\mathsf{ID}_2 + 2$ before they have received at least $\mathsf{ID}_2 + 2$ pulses from v_2. The last two of these pulses have to be sent by v_2 either at Line 7 or at Line 15. ◀

▶ **Lemma 14.** *Node v_2 eventually executes Line 10, i.e., it receives a pulse on port 1 and then another pulse on some port q. Moreover, the second pulse delivered arrives on port $q = 0$ and v_2 enters the rebalancing phase.*

Proof. Any node $v \neq v_2$ in the ring has an ID that is at least $\mathsf{ID}_2 + 2$. Consequently, by Lemma 13 and Lemma 12, every node eventually completes ID_2 iterations of the competing phase and every node $v \neq v_2$ enters iteration $\mathsf{ID}_2 + 1$. Thus, in particular, v_1 and v_3 send pulses to v_2 in iteration $\mathsf{ID}_2 + 1$. Consequently, v_2 will eventually receive a pulse from v_3 on port 1 (i.e., it will not be blocked at Line 8). Moreover, since v_1 has sent a pulse `p` to v_2 at iteration $\mathsf{ID}_2 + 1$, v_2 will eventually receive a pulse on some port q (i.e., it will execute Line 9). As long as v_2 does not send a pulse to v_3, v_3 will not complete iteration $\mathsf{ID}_2 + 1$ of the competing phase, and it will thus not send another pulse to v_2 before `p` is delivered. Consequently, $q = 0$ and v_2 enters the rebalancing phase. ◀

Notice that if v_2 is only a local minimum among v_1, v_2, and v_3, rather than a global one, it is still possible to adapt the proof to obtain a weaker variant of the lemma. In this variant, if v_2 executes Line 10, it must do so with $q = 0$.

From Lemma 14 we have:

▶ **Corollary 15.** *Node v_2 does not enter the global termination phase and thus it cannot terminate the algorithm in the* Leader *state.*

The following observation follows from the fact that the second minimum identifier in the ring is at least $\mathsf{ID}_2 + 2$. Recall that the reduced ring is obtained by removing node v_2 and directly connecting v_1 and v_3 while respecting the original port labeling.

▶ **Observation 16.** *When the algorithm is executed on the reduced ring, v_1 and v_3 send at least $\mathsf{ID}_2 + 2$ pulses to each other (and they receive these pulses) during their competing phase.*

▶ **Lemma 17.** *When node v_2 completes the rebalancing phase, v_2 has sent exactly $\mathsf{ID}_2 + 2$ pulses to node v_1 and $\mathsf{ID}_2 + 2$ pulses to node v_3. Moreover, at the end of the rebalancing phase, v_2 has received exactly $\mathsf{ID}_2 + 2$ pulses from v_1 and $\mathsf{ID}_2 + 2$ pulses from v_3.*

Proof. Once v_2 terminates its competing phase it has sent/received ID_2 pulses to/from each of its neighbors. At Line 7, it sends 2 pulses to v_1 and at Line 15 it sends 2 pulses to v_3. Moreover, at Lines 8 and 17, it has received two pulses from v_3 and at Lines 9 and 16 it has received two pulses from v_1 (recall, Lemma 14, that if Line 16 is executed then $q = 0$ at Line 9). ◀

▶ **Observation 18.** *From Observation 16 and Lemma 17, v_1 and v_3 cannot distinguish during the first $\mathsf{ID}_2 + 2$ iterations of their competing phase if they are in the original ring or in the reduced ring.*

We are now ready to prove Proposition 10.

Proof of Proposition 10. By Observation 18, before node v_2 completes its rebalancing phase, nodes v_1 and v_3 are not able to distinguish this execution from the one on the reduced ring. By Lemma 14, node v_2 completes its rebalancing phase and it eventually enters the relaying phase. It is easy to see that, from this point onward, v_2 behaves as an asynchronous link between v_1 and v_3 as long as v_2 does not exit the loop at Lines 19–23. ◀

From Proposition 10, it follows that as long as no node terminates and there are at least two active nodes, the one with the minimum identifier will enter the relaying phase, and the other active nodes will behave as if this node were not present in the ring.

Observe that the global minimality of ID_2 is used only in the proof of Lemma 14, to establish that v_2 must enter the relaying phase, all the other lemmas, when needed, they only require the identifier of v_2 to be a local minimum among v_1, v_2, v_3. As a matter of fact, a node v' may have an identifier that is a local minimum over a sufficiently long sequence of nodes, allowing it, in some executions, to enter the relaying phase before the global minimum. This does not affect the indistinguishability claimed in Proposition 10 that remains valid for v' and its two neighbours.

We now show that no node terminates before the one with the maximum identifier exits its competing phase. Moreover, we show that this node eventually enters the global termination phase, triggering the algorithm's termination with the election of the node with the maximum identifier. We will show that each node terminates when it has received three more pulses on one port than the other. In the next lemma, we characterize when such an event can happen.

▶ **Lemma 19.** *During the execution, if at some point, some node v has sent three more pulses on port q than it has sent on port $1-q$, or if v has sent three more pulses on port q than it has received on port q, then either v has executed Line 11, or v is a relaying node and v has previously received three more pulses on port $1-q$ than on port q.*

In any case, when this happens, the node v does not send any pulses afterwards.

Proof. During the competing phase, on each port, each node alternately sends a pulse on each port and receives a pulse on each port. At the end of the competing phase, node with identifier ID has sent and received ID pulses on each port. At Line 7, it sends two pulses on port 0. Then, if the condition at Line 10 is true, it sends a third pulse on port 0 (i.e., it executes Line 11) and stops sending pulses afterwards.

Suppose now that the condition at Line 10 is not satisfied. Then it means that at Line 9, $q = 0$ and at this point, v has sent $\mathsf{ID}+2$ pulses on port 0, has sent ID pulses on port 1, has received $\mathsf{ID}+1$ pulses on port 0, and has received $\mathsf{ID}+1$ pulses on port 1. After the rebalancing phase (Lines 15 to 17), v has sent and received $\mathsf{ID}+2$ pulses on each port.

After that, it just relays the pulses in the order they arrive and stores the difference between the number of pulses sent on each port in the variable Diff. Thus, after each iteration of the loop of the relaying phase, the number of pulses it has sent on a port q is precisely the number of pulses it has received on port $1-q$. Thus, it will send three more pulses on port q than on port $1-q$ only if it has received three more pulses on port $1-q$ than on port q. When this happens, $|\mathsf{Diff}| = 3$ and node v exits the loop and terminates. ◀

▶ **Corollary 20.** *If a node v terminates the algorithm (i.e., executes Line 13 or 24), then there is a node v' that has executed Line 11, and v' will eventually terminate in the* Leader *state.*

Proof. By contradiction, suppose that some node v terminates and that no node has executed Line 11 before. Then v necessarily terminates the algorithm at Line 24, and thus at some time t, v has received three more pulses on some port q than on port $1-q$. Among all such nodes, consider the node v such that this time t is minimal. Let v' be the neighbor of v behind port q. By Lemma 19, v is a relaying node, and v has received three more pulses from v' than it has sent to v'. Consequently, there exists a time $t' < t$ where v' has sent at least three more pulses to v than it has received from v. By Lemma 19 and since we assumed that v' has not executed Line 11, it implies that at time t', v' has received three more pulses on one port than on the other. But this contradicts the definition of v and t. ◀

By iteratively applying Corollary 15 and Proposition 10, and by observing that, by Corollary 20, no node terminates early, we obtain the following corollary:

▶ **Corollary 21.** *The node v with the maximum identifier eventually becomes the only competing node. Moreover, any other node eventually becomes a relaying node and cannot terminate the algorithm in the* Leader *state.*

We now show that the node with the maximum identifier eventually enters the Leader state and that, after this point, no more pulses are sent in the network, and every other node eventually enters the Non-Leader state.

▶ **Lemma 22.** *In a ring containing only one competing node v, the condition at Line 10 is true for v. Moreover, v will eventually execute Lines 11, 12, and 13, terminating the algorithm in the* Leader *state. Furthermore, when v executes Line 13, there are no pulses in transit and for every other node v', we have $|\mathsf{Diff}(v')| = 3$ and thus v' eventually terminates in the* Non-Leader *state.*

Proof. Suppose that there is a moment t where the ring contains only one competing node v. Then, by Proposition 10, Corollary 15, and Corollary 20 applied iteratively to all nodes except v, node v cannot distinguish the original ring from a ring containing only v.

Thus during the competing phase, v sends and receives $\mathsf{ID}(v)$ pulses to itself in each direction. In the solitude-checking phase, v sends two pulses on port 0 that are eventually delivered to itself on port 1. Once v has received both pulses, the condition at Line 10 is true and v enters the global termination phase. It sends another pulse to itself on port 0 (Line 11) that is eventually delivered on port 1 and v can then execute Lines 12 and 13 terminating in the Leader state.

Note that by Corollary 15, when v exits its competing phase, every other node v' is a relaying node. Consequently, at this point, it has sent and received the same number of pulses on each port by Observation 11. Therefore, at this point, there are no pulses in transit in the network and, by Lemma 17, for each node $v' \neq v$, we have $\mathsf{Diff}(v') = 0$.

Once v has executed Line 7, there are two pulses in transit in the network that are moving in the same direction and each node $v' \neq v$ relays them (through the same port) before they reach v on port 1. So when v executes Line 9, for each node $v' \neq v$, we have $|\mathsf{Diff}(v')| = 2$. Then, when v executes Line 11, it sends a third pulse in the same direction that will eventually reach v on port 1, at which point there are no pulses in transit. In the meantime, every node $v' \neq v$ will have relayed it and its variable $\mathsf{Diff}(v')$ will satisfy $|\mathsf{Diff}(v')| = 3$. Thus, after v' has relayed this third pulse, v' will exit the loop in the relaying phase and it will terminate in the Non-Leader state. ◀

Proof of Theorem 2. It suffices to show that Algorithm 1 is a correct leader election algorithm for all unoriented rings and it has a message complexity of $O(n\mathsf{ID}_{\max})$.

Correctness follows directly from Corollary 21 and Lemma 22. Regarding the message complexity, take the node $v_{\max}$ with maximum identifier $\mathsf{ID}_{\max}$, by Corollary 21 and Lemma 22, this node will execute Lines 1–13, sending $4\mathsf{ID}_{\max} + 3$ pulses (the factor 4 comes from the fact that we initially double each identifier and send pulses in both directions) in the process. Consider now any other node v_j with identifier $\mathsf{ID}_j \neq \mathsf{ID}_{\max}$, by Corollary 15 and Lemma 17, v_j first executes Lines 1–9 and Lines 15–17, the number of pulses sent at these lines is $4\mathsf{ID}_j + 4$, and then it executes Lines 19–23, but here it will only send a pulse when it receives one. Therefore, Lemma 22 and the fact that $v_{\max}$ sends $4\mathsf{ID}_{\max} + 3$ pulses show that v_j cannot send more than $\delta_j = (4\mathsf{ID}_{\max} + 3) - (4\mathsf{ID}_j + 4)$ pulses while relaying. The total number of pulses sent by Algorithm 1 is therefore at most $n(4\mathsf{ID}_{\max} + 3)$. ◀

5 Conclusions and Open Questions

In this paper, we demonstrate that non-uniform quiescently terminating content-oblivious leader election is achievable in general 2-edge-connected networks, and that uniform quiescently terminating content-oblivious leader election is achievable in unoriented rings.

Previously, such leader election was only known to be possible in ring topologies – specifically, a quiescently terminating algorithm in oriented rings and a stabilizing algorithm in unoriented rings [24]. Consequently, we remove the need for a preselected leader in the general algorithm simulation result by Censor-Hillel, Cohen, Gelles, and Sela [7], trading this requirement for the much weaker assumption of non-uniformity. For the special case of ring topologies, we provide a definitive answer on the computational equivalence of fully-defective communication and noiseless communication.

Several intriguing open questions remain. The foremost is whether *uniform* content-oblivious leader election algorithm exists for general 2-edge-connected topologies. So far, this has only been established for oriented rings [24] and later for unoriented rings (Theorem 2). Our current algorithm requires an upper bound N on the number of nodes n, as node identifiers must be spaced N apart to ensure that no node other than the leader r satisfies the Leader exit condition $\mathsf{Count}(v) = \mathsf{ID}(v)$. This reliance on N seems inherent to our approach, where counters of adjacent nodes may differ by one.

Another important open question is whether the efficiency of our algorithms can be improved. Our algorithm for general 2-edge-connected topologies is highly sequential due to its use of DFS. A common way to measure *time complexity* of an asynchronous algorithm is to assume that each message takes one unit of time to be delivered [3]. Under this model, the time complexity of our algorithm is $O(m \cdot N \cdot \mathsf{ID}_{\min})$, matching its message complexity, as the DFS traverses the m edges sequentially, and the traversal of each edge requires sending $O(N \cdot \mathsf{ID}_{\min})$ pulses. It remains unknown whether we can make the algorithm less sequential.

Much less is known about message complexity lower bounds for content-oblivious leader election. The only known result is a lower bound of $\Omega\left(n \cdot \log \frac{k}{n}\right)$, where k is the number of distinct, assignable identifiers in the network, due to Frei, Gelles, Ghazy, and Nolin [24]. Currently, there remains a significant gap between this lower bound and the upper bounds, both in rings and in general 2-edge-connected networks. A key open question is whether the message complexity for general 2-edge-connected networks is inherently higher than that of ring topologies. If so, how can we leverage the structural properties of the hard instances to establish such a lower bound?

References

1 Dan Alistarh, Joel Rybicki, and Sasha Voitovych. Near-optimal leader election in population protocols on graphs. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 246–256, 2022. doi:10.1145/3519270.3538435.

2 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006. doi:10.1007/S00446-005-0138-3.

3 Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004.

4 Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

5 Michael Barborak, Anton Dahbura, and Miroslaw Malek. The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSur)*, 25(2):171–220, 1993. doi:10.1145/152610.152612.

6 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 119–129, 2020. doi:10.1145/3357713.3384312.

7 Keren Censor-Hillel, Shir Cohen, Ran Gelles, and Gal Sela. Distributed computations in fully-defective networks. *Distributed Computing*, 36(4):501–528, 2023. doi:10.1007/S00446-023-00452-2.

8 Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. Making asynchronous distributed computations robust to noise. *Distributed Computing*, 32:405–421, 2019. doi:10.1007/S00446-018-0343-5.

9 Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou. Content-oblivious leader election in 2-edge-connected networks, 2025. doi:10.48550/arXiv.2507.08348.

10 Yi-Jun Chang, Tsvi Kopelowitz, Seth Pettie, Ruosong Wang, and Wei Zhan. Exponential separations in the energy complexity of leader election. *ACM Transactions on Algorithms*, 15(4), 2019. doi:10.1145/3341111.

11 Imrich Chlamtac and Shay Kutten. On broadcasting in radio networks-problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 2003.

12 Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pages 148–162, Berlin, Heidelberg, 2010. Springer-Verlag. doi:10.1007/978-3-642-15763-9_15.

13 Artur Czumaj and Peter Davies. Leader election in multi-hop radio networks. *Theor. Comput. Sci.*, 792(C):2–11, November 2019. doi:10.1016/j.tcs.2019.02.027.

14 Artur Czumaj and Peter Davies. Exploiting spontaneous transmissions for broadcasting and leader election in radio networks. *Journal of the ACM (JACM)*, 68(2):1–22, 2021. doi:10.1145/3446383.

15 Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982. doi:10.1016/0196-6774(82)90004-9.

16 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018. doi:10.1007/S00446-016-0281-Z.

17 Elena Dubrova. *Fault-Tolerant Design*. Springer, Berlin, 2013. doi:10.1007/978-1-4614-2113-9.

18 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Beeping a deterministic time-optimal leader election. In *32nd International Symposium on Distributed Computing (DISC 2018)*, pages 20:1–20:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.20.

19 Yuval Emek and Eyal Keren. A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 93–102, 2021. doi:10.1145/3465084.3467922.

20 Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing (PODC)*, pages 137–146, 2013. doi:10.1145/2484239.2484244.

21 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

22 Orr Fischer and Merav Parter. Distributed CONGEST algorithms against mobile adversaries. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 262–273. ACM, 2023. doi:10.1145/3583668.3594578.

23 Klaus-Tycho Förster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks. In *Proceedings of 28th International Symposium on Distributed Computing (DISC)*, pages 212–226. Springer, 2014.

24 Fabian Frei, Ran Gelles, Ahmed Ghazy, and Alexandre Nolin. Content-Oblivious Leader Election on Rings. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.26.

25 Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13(1–2):1–157, 2017. doi:10.1561/0400000079.

26 Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 748–766. SIAM, 2013. doi:10.1137/1.9781611973105.54.

27 Yael Hitron and Merav Parter. Broadcast CONGEST algorithms against adversarial edges. In Seth Gilbert, editor, *Proceedings of the 35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 23:1–23:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.23.

28 Yael Hitron and Merav Parter. General CONGEST compilers against adversarial edges. In Seth Gilbert, editor, *Proceedings of the 35th International Symposium on Distributed Computing (DISC)*, volume 209 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.24.

29 Yael Hitron, Merav Parter, and Eylon Yogev. Broadcast CONGEST algorithms against eavesdroppers. In Christian Scheideler, editor, *Proceedings of the 36th International Symposium on Distributed Computing (DISC)*, volume 246 of *LIPIcs*, pages 27:1–27:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.27.

30 Yael Hitron, Merav Parter, and Eylon Yogev. Secure distributed network optimization against eavesdroppers. In Yael Tauman Kalai, editor, *Proceedings of the 14th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 251 of *LIPIcs*, pages 71:1–71:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.71.

31 Abhishek Jain, Yael Tauman Kalai, and Allison Bishop Lewko. Interactive coding for multiparty protocols. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 1–10, 2015. doi:10.1145/2688073.2688109.

32 Israel Koren and C. Mani Krishna, editors. *Fault-Tolerant Systems*. Morgan Kaufmann, San Francisco, 2nd edition, 2020. doi:10.1016/B978-0-12-818105-8.

33 Merav Parter. A graph theoretic approach for resilient distributed algorithms. In Alessia Milani and Philipp Woelfel, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, page 324. ACM, 2022. doi:10.1145/3519270.3538453.

34 Merav Parter and Eylon Yogev. Distributed algorithms made secure: A graph theoretic approach. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1693–1710. SIAM, 2019. doi:10.1137/1.9781611975482.102.

35 Merav Parter and Eylon Yogev. Secure distributed computing made (nearly) optimal. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 107–116. ACM, 2019. doi:10.1145/3293611.3331620.

36 Andrzej Pelc. Reliable communication in networks with byzantine link failures. *Networks*, 22(5):441–459, 1992. doi:10.1002/NET.3230220503.

37 Sridhar Rajagopalan and Leonard Schulman. A coding theorem for distributed computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 790–799, 1994. doi:10.1145/195058.195462.

38 Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. Springer, Cham, 2018. doi:10.1007/978-3-319-94141-7.

39 Herbert Ellis Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.

40 Nicola Santoro and Peter Widmayer. Time is not a healer. In *STACS 1989*, volume 349 of *Lecture Notes in Comput. Sci.*, pages 304–313. Springer, 1989. doi:10.1007/BFB0028994.

41 Leonard J Schulman. Communication on noisy channels: A coding theorem for computation. In *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pages 724–733. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267778.

42 Leonard J Schulman. Deterministic coding for interactive communication. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 747–756, 1993. doi:10.1145/167088.167279.

43 Leonard J Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. doi:10.1109/18.556671.

44 Yuichi Sudo and Toshimitsu Masuzawa. Leader election requires logarithmic time in population protocols. *Parallel Processing Letters*, 30(01):2050005, 2020.

45 Robin Vacus and Isabella Ziccardi. Minimalist leader election under weak communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 406–416, 2025. doi:10.1145/3732772.3733559.

A Additional Related Work

Fault-tolerant distributed computing involves two main challenges. The first is message corruption, which can occur due to factors such as channel noise. The second is the presence of faulty nodes and edges, which can arise from issues like unreliable hardware or malicious attacks.

To address message corruption, a common approach is to introduce additional redundancy using coding techniques, known as *interactive coding*. The noise affecting the messages can be either random or adversarial, and there is usually an upper bound on the level of randomness or adversarial behavior. The study of interactive coding was initiated by Schulman [41, 42, 43] in the two-party setting and later extended to the multi-party case by Rajagopalan and Schulman [37]. For a comprehensive overview of interactive coding, see the survey by Gelles [25].

For networks with n nodes, the maximum fraction of corrupted messages that a distributed protocol can tolerate is $\Theta(1/n)$ [31]. If more than this fraction of messages are corrupted, the adversary can fully disrupt the communication of the node that communicates the fewest messages. Censor-Hillel, Gelles, and Haeupler [8] presented a distributed interactive coding scheme that simulates any asynchronous distributed protocol while tolerating an optimal corruption of $\Theta(1/n)$ of all messages. A key technique underlying their algorithm is a content-oblivious BFS algorithm.

The celebrated impossibility theorem of Fischer, Lynch, and Paterson [21] states that achieving consensus in an asynchronous distributed system is impossible for a deterministic algorithm when one or more nodes may crash. Similar impossibility results on the solvability of consensus have been found when processes are correct but may experience communication failures, such as message omissions, insertions, and corruptions [40].

If the number of Byzantine edges is bounded by f, reliable communication can only be achieved if the graph is at least $2f$-edge-connected [15, 36].

A series of recent studies has focused on resilient and secure distributed graph algorithms in the synchronous setting [35, 34, 28, 27, 29, 33, 22, 30]. These works have developed compilation schemes that transform standard distributed algorithms into resilient and secure versions. For instance, in a $(2f+1)$-edge-connected network, any distributed algorithm in the CONGEST model can be adapted to remain resilient against up to f adversarial edges [28].

Content-oblivious computation has also been studied in the synchronous setting. In the *beeping model* introduced by Cornejo and Kuhn [12], during each communication round, a node can either beep or listen. If the node listens, it can distinguish between silence or the presence of at least one beep. Several leader election algorithms have been developed for the beeping model and its variations [13, 18, 23, 26, 45].

Several additional distributed models have been designed with limited communication capabilities to capture various real-world constraints. These include radio networks [11], population protocols [2], and stone-age distributed computing [20]. Leader election has been extensively studied in these models [1, 6, 13, 14, 10, 16, 19, 26, 44, 45].

B Pseudocode for Theorem 1

In this appendix, we present the pseudocode for leader election in general 2-edge-connected networks. The algorithm for Theorem 1 is presented in Algorithm 2, with the DFS notification subroutine detailed in Algorithm 3. There are five main variables used in our algorithm:

State(v): This variable represents the current state of v. We say a node v is in the *synchronized counting* phase if $\mathsf{State}(v) = \perp$. Otherwise, the node is in the *DFS notification* phase, and $\mathsf{State}(v) \in \{\mathsf{Leader}, \mathsf{Non\text{-}Leader}\}$ indicates the output of v. At the start, every node is in the synchronized counting phase. Once a node advances to the DFS notification phase, it fixes its output and cannot go back to the synchronized counting phase.

LeaderID(v): This variable stores the identifier of the leader as known by node v.

Parent(v): This variable stores the port number through which v connects to its parent in T_G.

Count(v): This variable implements the counter used in the synchronized counting phase.

$\mathcal{P}(v)$: This variable represents the remaining unexplored ports of v in the DFS notification phase. Precisely, $\mathcal{P}(v)$ is the set of ports on which v has not yet received $\mathsf{LeaderID} + N + 2$ pulses.

For ease of presentation, we define the following two actions:

SendPulsesUntil$_i(k)$: Send pulses along port i until a total of k pulses have been sent from the start of the algorithm (i.e., $\sigma_i = k$).

SendAll(1): Send one pulse along each port.

Events. For ease of analysis, several positions in Algorithm 2 and Algorithm 3 are annotated with labels: $\mathsf{StartDFS}(v)$ (node v initiates the DFS), $\mathsf{SendExplore}_i(v)$ (node v begins sending an explore-notification on port i), $\mathsf{ReceiveExplore}_i(v)$ (node v confirms receipt of an explore-notification from port i), $\mathsf{SendDone}_i(v)$ (node v begins sending a done-notification on port i), and $\mathsf{ReceiveDone}_i(v)$ (node v confirms receipt of a done-notification from port i). These labels are treated as *events* in the analysis in the full version [9] of the paper.

Algorithm 2 Leader election algorithm for node v.

```
1  ID(v) ← ID(v) · N          // Forcing all IDs to be integer multiples of N
2  State ← ⊥; LeaderID ← ⊥; Parent ← ⊥
3  Count ← 0
4  SendAll(1)
5  while State = ⊥ do                                   // Synchronized counting
6      on event ρ_i is incremented for some i ∈ [deg(v)] do
7          if ρ_i = min_{j∈[deg(v)]} ρ_j then           // Counter increment condition
8              Count ← Count + 1
9              SendAll(1)
10             if Count = ID(v) then                    // Leader exit condition
11                 State ← Leader
12                 LeaderID ← ID(v)
13                 Event: StartDFS(v)
14         else if ρ_i − σ_i > 1 then                   // Non-Leader exit condition
15             State ← Non-Leader
16             Parent ← i
17             LeaderID ← ⌊Count/N⌋ · N
18             wait until ρ_i = LeaderID + N + 2
19             Event: ReceiveExplore_i(v)
20     end
21 end
22 Notify(State, LeaderID, Parent)                      // Algorithm 3
```

Algorithm 3 Notify(State, LeaderID, Parent) for node v.

```
1  P = [deg(v)]                                              // The set of all ports
2  if State = Non-Leader then
3  |  P ← P \ {Parent}
4  while P ≠ ∅ do
5  |  j ← min(P)    // Traversal order:  prioritizing smaller port numbers
6  |  Event: SendExplore_j(v)
7  |  wait until ρ_j ≥ LeaderID + 1
8  |  SendPulsesUntil_j(LeaderID + N + 2)                   // Continue DFS on port j
9  |  while j ∈ P do
10 |  |  on event ρ_h = LeaderID + N + 2 for some h ∈ P do
11 |  |  |  if j = h then                                   // DFS returned from port j
12 |  |  |  |  P ← P \ {j}
13 |  |  |  |  Event: ReceiveDone_j(v)
14 |  |  |  else                  // Incoming DFS from port h via a back edge
15 |  |  |  |  P ← P \ {h}
16 |  |  |  |  Event: ReceiveExplore_h(v)
17 |  |  |  |  Event: SendDone_h(v)
18 |  |  |  |  SendPulsesUntil_h(LeaderID + N + 2)
19 |  |  end
20 |  end
21 end
22 if State = Non-Leader then
23 |  Event: SendDone_Parent(v)
24 |  SendPulsesUntil_Parent(LeaderID + N + 2)          // Return DFS on parent port
```

The Complexity Landscape of Dynamic Distributed Subgraph Finding

Yi-Jun Chang
National University of Singapore, Singapore

Lyuting Chen
National University of Singapore, Singapore

Yanyu Chen
National University of Singapore, Singapore

Gopinath Mishra
National University of Singapore, Singapore

Mingyang Yang
National University of Singapore, Singapore

Abstract

Bonne and Censor-Hillel (ICALP 2019) initiated the study of distributed subgraph finding in dynamic networks of limited bandwidth. For the case where the target subgraph is a clique, they determined the tight bandwidth complexity bounds in nearly all settings. However, several open questions remain, and very little is known about finding subgraphs beyond cliques. In this work, we consider these questions and explore subgraphs beyond cliques in the *deterministic* setting.

For finding cliques, we establish an $\Omega(\log\log n)$ bandwidth lower bound for one-round membership-detection under edge insertions only and an $\Omega(\log\log\log n)$ bandwidth lower bound for one-round detection under both edge insertions and node insertions. Moreover, we demonstrate new algorithms to show that our lower bounds are *tight* in bounded-degree networks when the target subgraph is a triangle. Prior to our work, no lower bounds were known for these problems.

For finding subgraphs beyond cliques, we present a *complete characterization* of the bandwidth complexity of the membership-listing problem for every target subgraph, every number of rounds, and every type of topological change: node insertions, node deletions, edge insertions, and edge deletions. We also show partial characterizations for one-round membership-detection and listing.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, dynamic algorithms, subgraph finding

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.22

Related Version *Full Version*: `https://arxiv.org/abs/2411.11544` [15]

Funding The research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (24-1323-A0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

1 Introduction

Detecting small subgraphs in distributed networks has recently attracted significant research interest [6, 9, 12, 16, 17, 20, 22, 25, 26, 27, 33, 30, 36]. Distributed subgraph finding plays an important role in understanding the *bandwidth limitation* in distributed networks: It is a classical problem where a straightforward $O(1)$-round solution exists with unlimited bandwidth, but becomes significantly more complex when bandwidth constraints are imposed.


© Yi-Jun Chang, Lyuting Chen, Yanyu Chen, Gopinath Mishra, and Mingyang Yang;
licensed under Creative Commons License CC-BY 4.0



Previous works on distributed subgraph findings mostly assumed a model in which the underlying network is *static*. However, distributed systems in real life may undergo topological changes over time: A node might crash, and a new connection might be formed between two existing nodes. To address this gap, Bonne and Censor-Hillel [6] initiated the study of distributed subgraph finding in dynamic networks to better capture the real-world behavior of networks of limited bandwidth. For the case where the target subgraph is a clique, they determined the tight bandwidth complexity bounds in nearly all settings. Later, Liu [36] extended this study to dynamic graphs with batched updates and resolved an open question of Bonne and Censor-Hillel [6, Open question 4]. However, several open questions remain, and very little is known about finding subgraphs beyond cliques. In this paper, we build upon their works [6, 36] to consider the remaining open questions and explore other target subgraphs beyond cliques.

1.1 Models

We now formally describe the models considered in this paper, which were introduced by Bonne and Censor-Hillel [6]. A dynamic network $\mathcal{G}$ is a sequence of graphs $\mathcal{G} = \{G^0, G^1, \ldots\}$. The superscript notation should not be confused with graph powers. The initial graph G^0 represents the state of the network at some starting point. For each $i > 1$, the graph G^i is either identical to its preceding graph G^{i-1} or obtained from G^{i-1} by a single topological change.

Each node u in the network is equipped with a unique identifier $\mathrm{ID}(u)$, and it knows the list of identifiers of all its neighbors. The communication is synchronous. In each round of communication, each node can send to each of its neighbors a message of B bits, where B denotes the *bandwidth* of the network.

We assume that each node initially knows the *entire* topology of the initial graph G^0. In particular, the set of all identifiers is global knowledge, so we may assume that the range of the identifiers is exactly $[n]$, where n is the number of nodes in the network.

Topological changes. We consider four types of topological changes: node insertions, node deletions, edge insertions, and edge deletions. In the case of a node insertion, the adversary may connect the new node to an arbitrary subset of the existing nodes. Each node u can only *indirectly* deduce that a topological change has occurred by comparing its list of neighbors $N_{G^i}(u)$ in the current round i and its list of neighbors $N_{G^{i-1}}(u)$ in the previous round $i-1$. At most one topological change can occur in each round. Suppose at some round i, node u detects that exactly one new node v appears in its neighborhood list, then from this information only, node u cannot distinguish whether edge $\{u, v\}$ is added or node v is added in round i, if we are in the model where both edge insertions and node insertions are allowed. Similarly, suppose node u detects that exactly one node v disappears in its list. In that case, u cannot distinguish whether edge $\{u, v\}$ is deleted or node v is deleted, if we are in the model where both edge deletions and node deletions are allowed.

Algorithms. An algorithm can be designed to handle only one type of topological change or any combination of them. Throughout this paper, we only consider *deterministic* algorithms. We say an algorithm $\mathcal{A}$ is an r-round algorithm if $\mathcal{A}$ works in the following setting.

- Each topological change is followed with at least $r-1$ *quiet rounds*. Specifically, if a topological change occurs in round i, then we must have $G^i = G^{i+1} = \cdots = G^{i+r-1}$. Rounds $i+1, \ldots, i+r-1$ are quiet in the sense that no topological changes occur in these rounds.
- The output w.r.t. G^i must be computed correctly by round $i+r-1$.

A one-round algorithm is not called a zero-round algorithm because, within each round, topological change occurs before any communication takes place. This setup allows the nodes to have one round of communication between the topological change and deciding the output within a single round. For example, if an edge $\{u, v\}$ is inserted in a round, then u and v can immediately communicate with each other along this edge within the same round.

We emphasize that all our algorithms and lower bounds in this paper are *deterministic*, although many of our lower bounds also extend to the randomized setting, as shown in the full version [15] of the paper.

1.2 Problems

We consider the four types of distributed subgraph finding problems introduced by Bonne and Censor-Hillel [6].

Membership Listing For the membership-listing ($\textsc{MemList}(H)$) problem, each node v lists all the copies of H containing v. In other words, for each copy of H and each node u in H, node u lists H.

Membership Detection For the membership-detection ($\textsc{MemDetect}(H)$) problem, each node v decides whether v belongs to at least one copy of H.

Listing For the listing ($\textsc{List}(H)$) problem, every appearance of H is listed by at least one node in the network. In other words, for each copy of H, there exists some node u that lists H.

Detection For the detection ($\textsc{Detect}(H)$) problem, the existence of any H must be detected by at least one node. Specifically, if the network does not contain H as a subgraph, then all nodes must output No. Otherwise, at least one node must output Yes.

For both membership-detection and detection, the output of each node is binary (Yes/No) only, with no requirement to report the actual target subgraph. For both membership-listing and listing, each node outputs a list of the target subgraphs, using the node identifiers in the network. For the listing problem, the node u responsible for listing H is not required to be in H, and it is allowed that each copy of H is listed by multiple nodes.

In the literature, the problem of deciding whether a subgraph isomorphic to H exists is often referred to as *H-freeness*, whereas *detection* sometimes denotes the task of outputting a copy of H. We emphasize that our use of detection differs from this convention: In both $\textsc{MemDetect}(H)$ and $\textsc{Detect}(H)$, outputting a copy of H is not required.

Bandwidth complexities. The r-round *bandwidth complexity* of a problem is defined as the minimum bandwidth B for which there exists an r-round algorithm solving that problem with bandwidth B. Fix a target subgraph H, a round number r, and some type of topological changes. Let $B_{\textsc{MemList}}$, $B_{\textsc{MemDetect}}$, $B_{\textsc{List}}$, and $B_{\textsc{Detect}}$ denote the r-round bandwidth complexities of $\textsc{MemList}(H)$, $\textsc{MemDetect}(H)$, $\textsc{List}(H)$, and $\textsc{Detect}(H)$, respectively. The following observations were made by Bonne and Censor-Hillel [6].

▶ **Observation 1.1** ([6]). *Given any target subgraph H and any integer r,*

$$B_{\textsc{Detect}} \leq B_{\textsc{List}} \leq B_{\textsc{MemList}}$$

under any type of topological change.

▶ **Observation 1.2** ([6]). *Given any target subgraph H and any integer r,*

$$B_{\textsc{Detect}} \leq B_{\textsc{MemDetect}} \leq B_{\textsc{MemList}}$$

under any type of topological change.

Nontrivial target subgraphs. In this paper, we only focus on *nontrivial* target subgraphs H, meaning that we implicitly assume that H is connected and contains at least three nodes:

- If H is not connected, then all four problems are trivially impossible to solve, as we allow the network to be disconnected.
- If H is connected with exactly two nodes, then all four problems are trivially solvable with zero communication.

For example, when we say that the one-round bandwidth complexity of $\textsc{MemDetect}(K_s)$ under edge insertions is $\Omega(\log\log n)$, we implicitly assume that $s \geq 3$.

1.3 Our Contributions

While Bonne and Censor-Hillel [6] settled most of the bandwidth complexity bounds for clique finding, they left five open questions, one of which [6, Open question 4] was resolved by Liu [36]. In this paper, we investigate the remaining ones.

Finding cliques under edge insertions. In [6, Open question 1] and [6, Open question 3], Bonne and Censor-Hillel asked for the tight bound on the one-round bandwidth complexity of membership-detection for triangles and larger cliques under *edge insertions* only. For triangles, they obtained two upper bounds $O(\log n)$ and $O(\sqrt{\Delta \log n})$ [6]. For larger cliques, they obtained an upper bound $O(\sqrt{n})$ which works even for the membership-listing problem [6]. In this work, we show that these problems admit a bandwidth lower bound of $\Omega(\log\log n)$, which holds even in bounded-degree networks.

▶ **Theorem 1.3.** *For any constant $s \geq 3$, the one-round bandwidth complexity of $\textsc{MemDetect}(K_s)$ under edge insertions is $\Omega(\log\log n)$, even in bounded-degree dynamic networks.*

Prior to our work, no lower bound was known for this problem. Moreover, we complement our lower bound with a new $O(\log\log n)$-bandwidth triangle finding algorithm in bounded-degree networks, which is capable of solving the more challenging problem of membership-listing.

▶ **Theorem 1.4.** *There exists a one-round algorithm of $\textsc{MemList}(K_3)$ under edge insertions with bandwidth $O(\Delta^2 \log\log n)$, where Δ is the maximum degree of the dynamic network.*

Combining Theorem 1.3 and Theorem 1.4, we obtain a tight bound of membership-detection for triangles in bounded-degree dynamic networks.

▶ **Corollary 1.5.** *The one-round bandwidth complexity of $\textsc{MemDetect}(K_3)$ under edge insertions is $\Theta(\log\log n)$ in bounded-degree dynamic networks.*

Finding cliques under edge insertions and node insertions. The remaining two open questions of Bonne and Censor-Hillel [6] considered the model where two types of topological changes are allowed. Specifically, [6, Open question 2] and [6, Open question 5] asked for the tight bound on the one-round bandwidth complexity of the listing problem for triangles and larger cliques under both *edge insertions* and *node insertions*. Previously, these problems were known to have an upper bound of $O(\log n)$ [6]. In this work, we show that these problems admit a bandwidth lower bound of $\Omega(\log\log\log n)$, which applies to the easier problem of detection and holds in bounded-degree networks.

▶ **Theorem 1.6.** *For any constant $s \geq 3$, the one-round bandwidth complexity of* $\textsc{Detect}(K_s)$ *under both node insertions and edge insertions is* $\Omega(\log\log\log n)$*, even in bounded-degree dynamic networks.*

Same as Theorem 1.3, prior to our work, no lower bound was known for this problem. Interestingly, we are also able to match this lower bound with a new $O(\log\log\log n)$-bandwidth algorithm for listing triangles in bounded-degree networks.

▶ **Theorem 1.7.** *There exists a one-round algorithm of* $\textsc{List}(K_3)$ *under edge insertions and node insertions with bandwidth* $O(\Delta\log\log\log n)$*, where* Δ *is the maximum degree of the dynamic network.*

Combining Theorem 1.6 and Theorem 1.7, we obtain a tight bound for triangle detection in bounded-degree dynamic networks.

▶ **Corollary 1.8.** *The one-round bandwidth complexity of* $\textsc{Detect}(K_3)$ *under edge insertions and node insertions is* $\Theta(\log\log\log n)$ *in bounded-degree dynamic networks.*

Beyond cliques. It appears to be a challenging task to extend the current results beyond cliques. In the static setting, while the round complexity for k-clique listing has been settled [9, 12, 16, 17, 22, 30], far less is known about target subgraphs that are not cliques. In the dynamic setting, Bonne and Censor-Hillel [6] highlighted that cliques are *unique* in that they can be found trivially in *one round* if bandwidth is unrestricted.

In this work, we demonstrate that it is possible to achieve meaningful results beyond cliques, despite the inherent difficulties. For subgraph finding beyond cliques, we present a *complete characterization* of the bandwidth complexity for the membership-listing problem across all target subgraphs, all numbers of rounds, and all four types of topological changes: node insertions, node deletions, edge insertions, and edge deletions (Table 1). Moreover, we show partial characterizations for one-round membership-detection (Table 2) and listing (Table 3).

Our contribution lies in finding structures in the apparent chaos: We identify relevant graph classes (e.g., complete multipartite graphs) and parameters (e.g., node-edge versions of distance, radius, and diameter) in the area of dynamic distributed subgraph finding. For the cases where a full characterization has yet to be achieved, we identify remaining challenging open problems and outline future research directions. Addressing these challenges will likely require developing novel techniques.

1.4 Additional Related Work

König and Wattenhofer [32] conducted a systematic study of classical network problems under various types of topological changes, such as node or edge insertions and deletions. They said that a combination of a problem and a topological change is *locally fixable* if an existing solution can be repaired in $O(1)$ rounds following a topological change in the network. Several subsequent studies have investigated dynamic distributed algorithms for local problems, such as independent set [3, 13], matching [39], and coloring [37]. The dynamic-LOCAL model was formalized in [1]. Distributed algorithms for highly dynamic networks, where multiple topological changes can occur within a single communication round, were examined in [4, 10]. Global distributed problems, such as consensus and information dissemination, have also been studied in the dynamic setting [18, 21, 28, 31, 34, 35, 40]. Dynamic distributed algorithms are closely related to the concept of *self-stabilization* – a key notion in distributed computing – where a distributed network undergoes various changes and must rapidly return to a stable state after some quiet time [19].

There is a long line of research studying distributed subgraph finding in the CONGEST model. The first breakthrough in triangle detection was achieved by Izumi and Le Gall [30], who demonstrated that triangle detection and listing can be completed in $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$ rounds, respectively. After a series of work [9, 12, 14, 16, 17, 22], it was shown that $\tilde{\Theta}(n^{1-2/k})$ is the tight bound for k-clique listing via the use of expander decomposition and routing. Distributed cycle findings have also been extensively studied [20, 22, 25, 33]. Property-testing variants of the distributed subgraph finding problem have been explored [7, 11, 24, 26, 27]. The subgraph finding problem has also been investigated in other computational models beyond distributed and parallel computing [2, 5, 23]. For more on the distributed subgraph finding problem, see the survey by Censor-Hillel [8].

The study of distributed subgraph finding is partially motivated by the fact that many algorithms can be significantly improved if the underlying network does not contain certain small subgraphs. For instance, Pettie and Su showed distributed coloring algorithms for triangle-free graphs using fewer colors and rounds [38]. Similarly, Hirvonen, Rybicki, Schmid, and Suomela developed an efficient distributed algorithm to find large cuts in triangle-free graphs [29].

1.5 Roadmap

In Section 2, we review essential graph terminology and parameters and demonstrate how certain graph parameters determine the minimum number of rounds required for certain subgraph-finding tasks. In Section 3, we present a technical overview of our proofs. In Section 4, we conclude the paper with some open questions. In Section A, we provide tables that summarize our results for subgraph finding beyond cliques.

2 Preliminaries

In this section, we present the basic definitions used in the paper. In Section 2.1, we review essential graph terminology and parameters. In Section 2.2, we discuss some basic properties of these graph parameters. In Section 2.3, we demonstrate how specific graph parameters determine the minimum number of rounds required for MemList(H) and MemDetect(H), independent of any bandwidth constraints.

2.1 Graph Terminology

Given a graph H, an edge $e = \{u, v\} \in E(H)$, and a node subset $S \subseteq V(H)$, we write $H[S]$ to denote the subgraph of H induced by node set S, write $H - S$ to denote the subgraph of H induced by node set $V(H) \setminus S$, and write $H - e$ to denote the subgraph of H induced by edge set $E(H) \setminus \{e\}$.

For a graph $G = (V, E)$, we have the following definitions.

▶ **Definition 2.1** (Eccentricity, diameter, radius, and center)**.**

Eccentricity of u:	$\mathrm{ecc}_G(u) = \max\{\mathrm{dist}_G(u, v) : v \in V\}$
Diameter of G:	$\mathrm{diam}(G) = \max\{\mathrm{ecc}_G(u) : u \in V\}$
Radius of G:	$\mathrm{rad}(G) = \min\{\mathrm{ecc}_G(u) : u \in V\}$
Center of G:	$\mathrm{center}(G) = \{u \in V : \mathrm{ecc}_G(u) = \mathrm{rad}(G)\}$

In other words, the eccentricity of a node is the maximum of its distance to other nodes, the diameter of a graph is the maximum eccentricity of its nodes, the radius of a graph is the minimum eccentricity of its nodes, and the center of a graph is a node subset containing all nodes with eccentricity equal to the radius.

We consider the "node-edge" version of the definitions above.

▶ **Definition 2.2** (Node-edge distance). *For any $u \in V$ and any $e = \{v, w\} \in E$, the node-edge distance between them is defined as* $\mathrm{dist}_G(u, e) := 1 + \min\{\mathrm{dist}_G(u, w), \mathrm{dist}_G(u, v)\}$.

We overload the notation $\mathrm{dist}(\cdot, \cdot)$ since it is easy to distinguish $\mathrm{dist}(u, v)$ and $\mathrm{dist}(u, e)$.

▶ **Definition 2.3** (Node-edge eccentricity, diameter, radius, and center).

$$\begin{aligned}
&\textit{Node-edge eccentricity of } u: && \widetilde{\mathrm{ecc}}_G(u) = \max\{\mathrm{dist}_G(u, e) : e \in E\} \\
&\textit{Node-edge diameter of } G: && \widetilde{\mathrm{diam}}(G) = \max\{\widetilde{\mathrm{ecc}}_G(u) : u \in V\} \\
& && \quad = \max_{u \in V} \max_{e \in E} \mathrm{dist}_G(u, e) \\
&\textit{Node-edge radius of } G: && \widetilde{\mathrm{rad}}(G) = \min\{\widetilde{\mathrm{ecc}}_G(u) : u \in V\} \\
& && \quad = \min_{u \in V} \max_{e \in E} \mathrm{dist}_G(u, e) \\
&\textit{Node-edge center of } G: && \widetilde{\mathrm{center}}(G) = \{u \in V : \widetilde{\mathrm{ecc}}_G(u) = \widetilde{\mathrm{rad}}(G)\} \\
& && \quad = \{u \in V : \max_{e \in E} \mathrm{dist}_G(u, e) = \widetilde{\mathrm{rad}}(G)\}
\end{aligned}$$

The node-edge distance definition is needed to capture the complexity bounds for various problems discussed in this paper. For example, in C_4, any node can detect an edge deletion within one round, whereas in C_5, the node opposite the deleted edge cannot detect the topological change in a single round. This is reflected by the fact that $\widetilde{\mathrm{rad}}(C_4) = 2 < 3 = \widetilde{\mathrm{rad}}(C_5)$, while the standard radius definition $\mathrm{rad}(C_4) = 2 = \mathrm{rad}(C_5)$ is insufficient to capture the difference.

2.2 Properties of Graph Parameters

These distance-based parameters and their node-edge versions are closely related. Specifically, we have the following observation.

▶ **Observation 2.4.** *For any connected graph H,* $\mathrm{diam}(H) \le \widetilde{\mathrm{diam}}(H) \le \mathrm{diam}(H) + 1$.

Proof. Since for any node u and any edge $\{v, w\}$, $|\mathrm{dist}_H(u, w) - \mathrm{dist}_H(u, v)| \le 1$, the observation follows directly from the definition of diam and $\widetilde{\mathrm{diam}}$.

$$\begin{aligned}
\mathrm{diam}(H) &= \max_{u \in V} \max_{v \in V} \mathrm{dist}_H(u, v) \\
&\le \max_{u \in V} \max_{\{v,w\} \in E} 1 + \min\{\mathrm{dist}_H(u, w), \mathrm{dist}_H(u, v)\} = \widetilde{\mathrm{diam}}(H) \\
&\le 1 + \max_{u \in V} \max_{v \in V} \mathrm{dist}_H(u, v) = \mathrm{diam}(H) + 1.
\end{aligned}$$

◀

The class of *complete multipartite graphs* plays a crucial role in the study of the complexity landscape of dynamic distributed subgraph finding, as it can be characterized in terms of node-edge diameter or node-edge independence.

▶ **Definition 2.5** (Complete multipartite graphs). *A graph G is complete k-partite if its node set $V(G)$ can be partitioned into k independent sets $S_1, S_2, \ldots, S_k$ such that for any two nodes $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and only if $u \in S_i$ and $v \in S_j$ for some $i \neq j$. For any k, a complete k-partite graph is called a complete multipartite graph.*

▶ **Definition 2.6** (Node-edge independence). *In a graph G, a node w and an edge $\{u, v\}$ are independent if w is adjacent to neither u nor v in G. A graph G is node-edge independent if at least one of its node-edge pairs are independent.*

In other words, a graph G is node-edge independent if and only if it contains a co-P_3 (complement of a path with three nodes) as an induced subgraph. See Figure 1.

▶ **Lemma 2.7.** *A graph G has no induced P_3 if and only if its connected components are cliques.*

Proof. If every connected component of G is a clique, then any three nodes selected cannot induce a P_3. Conversely, suppose a graph G has no induced P_3 and there is a connected component that is not a clique, then there is a pair of disconnected nodes u and v in the same component. The two nodes u and v must be connected by some minimum-length path P_{uv} whose length is at least 2, since they are in the same connected component. Any three consecutive nodes on the path induce a P_3, which is a contradiction. ◀

▶ **Lemma 2.8.** *A graph G is complete multipartite if and only if it is* not *node-edge independent.*

Proof. Observe that a graph is complete multipartite if and only if its complement is a disjoint union of cliques. Hence Lemma 2.7 implies that a graph G is *not* node-edge independent $\iff$ G does not contain a co-P_3 as an induced subgraph $\iff$ complement of G is P_3-free $\iff$ G is complete multipartite. ◀

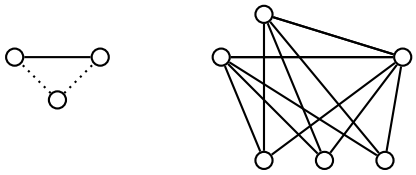


Figure 1 Co-P_3 (left, dotted lines denote non-edges) and a complete multi-partite graph (right).

Now we characterize complete multipartite graphs in terms of node-edge diameter.

▶ **Observation 2.9.** *For any connected graph H with at least three nodes, $\widetilde{\mathrm{diam}}(H) = 2$ if and only if H is complete multipartite.*

Proof. If $\widetilde{\mathrm{diam}}(H) = 2$, then H is not node-edge independent, since if node w is independent of edge $e = \{u, v\}$, then $\mathrm{dist}_H(w, e) \geq 3$. Hence, H is complete multipartite by Lemma 2.8. Conversely, suppose H is complete multipartite, then it is not node-edge independent, and $\mathrm{dist}_H(w, e) \leq 2$ for all $w \in V(H)$ and $e \in E(H)$. Hence, $\widetilde{\mathrm{diam}}(H) \leq 2$. Since H has at least three nodes, for any edge e, there is a node w such that $w \notin e$. We have $\mathrm{dist}_H(w, e) \geq 2$ and $\widetilde{\mathrm{diam}}(H) \geq 2$. ◀

2.3 Locality Constraints

In this section, we investigate the minimum number of rounds required for the two problems MemList(H) and MemDetect(H) regardless of any bandwidth constraints, for any given target subgraph H. Due to the *local* nature of topological changes, the appearance or disappearance of a copy of H might not be detectable for some nodes in H in the same round or even after several rounds of communication.

See Figure 2 for an example. Suppose at some round r, edge $\{u, v\}$ is inserted, so the current graph is C_5. Assuming unlimited bandwidth, after one round of messaging, all the nodes highlighted can detect the appearance of C_5 due to the messages from u or v. The only remaining node w is unable to detect the appearance of C_5 after one round of messaging, as the information about the topological change has not reached w.



Figure 2 Locality constraint for MemDetect(C_5).

In the subsequent discussion, we define two graph parameters r_H and r'_H, and use them to show the locality constraints for MemList(H) and MemDetect(H).

First, we define threshold r_H and show that r_H is a lower bound on the number of rounds needed for MemList(H) or MemDetect(H) under edge insertions or under edge deletions.

▶ **Definition 2.10** (Threshold r_H). *For any given graph H, define $r_H = \widetilde{\text{diam}}(H) - 1$.*

▶ **Theorem 2.11** (Locality constraints, edge insertions/deletions). *Given any graph H, for any $T < r_H$, there exists no T-round algorithm for* MemList(H) *or* MemDetect(H) *under edge insertions or under edge deletions.*

Proof. Suppose $\text{dist}_H(w, e) = \widetilde{\text{diam}}(H)$, for a node $w \in V(H)$ and an edge $e = \{u, u'\} \in E(H)$. Consider the following dynamic graph $\mathcal{G}$ under edge insertions:

1. Initially, $G^0 = H - e$.
2. At round 1, consider two cases:
   a. The edge $\{u, u'\}$ is inserted. Thus $G^1 = H$.
   b. There is no change. Thus G^1 does not contain any subgraph isomorphic to H.

Since $T < r_H = \widetilde{\text{diam}}(H) - 1 = \min\{\text{dist}_H(w, u'), \text{dist}_H(w, u)\}$, within T rounds of communication, node w must receive identical messages in both cases and cannot distinguish two cases, so any correct algorithm requires at least r_H rounds.

A similar proof applies to the case of edge deletion. The only required modification is to start with $G^0 = H$ and then replace the insertion of e with the deletion of e. ◀

Next, we define the threshold r'_H and show that r'_H is a lower bound on the number of rounds needed for MemList(H) or MemDetect(H) under node insertions or under node deletions.

▶ **Definition 2.12** (Threshold r'_H). *For any given graph H, define $r'_H = \text{diam}(H) - 1$.*

▶ **Theorem 2.13** (Locality constraints, node insertions/deletions). *Given any graph H, for any $T < r'_H$, there exists no T-round algorithm for* MemList(H) *or* MemDetect(H) *under node insertions or under node deletions.*

Proof. Suppose $\text{dist}_H(u, w) = \text{diam}(H)$ for nodes $u, w \in V(H)$. For node insertions, we start with the graph $G^0 = H - \{u\}$, the subgraph of H induced by $V(H) \setminus \{u\}$. Consider two cases:

1. Insert node u, along with all its incident edges in H. Thus $G^1 = H$.
2. No topological change.

Since $T < r'_H = \text{diam}(H) - 1 = \min\{\text{dist}_H(w, u') : u' \in N_H(u)\}$, node w receives the same information within T rounds in both cases. Therefore, for any T-round algorithm, w is unable to correctly decide whether H appears. We can use a similar design to prove the case of node deletions: Start with graph H, and then at round 1, either delete node u or do nothing. Since $T < r'_H = \min\{\text{dist}_H(w, u') : u' \in N_H(u)\}$, the same analysis shows that node w cannot decide whether H disappears. ◀

We show a tighter bound in terms of r_H for the case of node insertions. We remark that the same argument does work for node deletion since nodes are assumed to know the entire topology before any deletion and hence may decide based on one-sided information along the shortest path from u to x in the following example.

▶ **Theorem 2.14** (Locality constraints, node insertions). *Given any graph H, for any $T < r_H$, there exists no T-round algorithm for* MemList(H) *or* MemDetect(H) *under node insertions.*

Proof. According to Observation 2.4, we have either $r_H = \widetilde{\text{diam}}(H) - 1 = \text{diam}(H) - 1 = r'_H$ or $r_H = \widetilde{\text{diam}}(H) - 1 = \text{diam}(H) = r'_H + 1$. If $r_H = r'_H$, then the proof follows from Theorem 2.13. For the rest of the proof, we focus on the case where $r_H = \widetilde{\text{diam}}(H) - 1 = \text{diam}(H) = r'_H + 1$.

Suppose $\text{dist}_H(x, e) = \widetilde{\text{diam}}(H) = \text{diam}(H) + 1$ for node $x \in V(H)$ and edge $e = \{u, v\} \in E(H)$. We must have $\text{dist}_H(x, u) = \text{dist}_H(x, v) = \text{diam}(H)$. Otherwise, if $\text{dist}_H(x, u) \leq \text{diam}(H) - 1$, then $\text{dist}_H(x, e) \leq \text{diam}(H)$, contradicting our assumption. Now, consider the following dynamic graph $\mathcal{G}$.

1. Initially $G^0 = H - \{u\}$.
2. At round 1, consider two cases:
   a. Node u is inserted with all its incident edges in H. Thus $G^1 = H$.
   b. Node u is inserted with all its incident edges in H excluding $\{u, v\}$. Thus $G^1 = H - e$.

For any $T < r_H = \text{diam}(H)$, node x receives the same information within T rounds in both cases. Therefore, for any T-round algorithm, x cannot distinguish the two cases and cannot output correctly. ◀

One-round solvability and complete multipartite graphs. By Theorems 2.11 and 2.14 with $T = 1$, we know that $r_H \leq 1$, or equivalently $\widetilde{\text{diam}}(H) \leq 2$, characterizes the class of target subgraphs H that permits one-round MemList(H) and MemDetect(H) algorithms for edge insertions, edge deletions, and node insertions. Observe that $\widetilde{\text{diam}}(H) = 1$ if and only if H is a single edge, so all non-trivial target subgraphs H satisfy $\widetilde{\text{diam}}(H) \geq 2$. Therefore, Observation 2.9 implies that, excluding trivial target subgraphs, complete multipartite graphs are exactly the class of graphs that permits one-round MemList(H) and MemDetect(H) algorithms for edge insertions, edge deletions, and node insertions.

3 Technical Overview

In this section, we present a technical overview of our proofs.

3.1 Lower Bounds for Finding Cliques

We start with the lower bounds for finding cliques.

$\Omega(\log\log n)$ lower bound. We present the core idea underlying the $\Omega(\log\log n)$ bandwidth lower bound for one-round MemDetect(K_s) under edge insertions, as shown in Theorem 1.3. For simplicity and clarity, we focus on the case of membership-detecting triangles (K_3).

We start by describing the hard instance. Consider a set of $t = \log^{0.1} n$ nodes $\{x_1, x_2, \ldots, x_t\}$ and an independent set I. By connecting each x_i to two distinct nodes from I, we form t disjoint paths of length two, each resembling a triangle missing one edge. The graph is constructed by edge insertions over $2t$ rounds.

After constructing these paths, a single edge e is inserted. Two scenarios are possible:

- Edge e connects the two endpoints of one of the constructed paths, completing a triangle.
- Edge e connects endpoints from two different paths, forming no triangle.

Let (a, x_i, b) be one such path, and suppose a is one endpoint of the newly inserted edge e. To correctly detect whether a triangle has been formed, node a must determine whether the other endpoint of e is node b. We show that if the bandwidth of the algorithm is $B = o(\log\log n)$, then no mechanism exists for a to reliably make this distinction.

- Before the insertion of e, node a might try to learn ID(b) through x_i during the initial $2t$ rounds. However, due to bandwidth limitations, a can receive only $O(Bt) = o(\log n)$ bits, which is far too little to uniquely identify b from the space of possible identifiers $[n]$.
- After the insertion of e, the two endpoints of e can exchange B-bit messages. However, distinguishing whether they are part of the same path among t candidates requires $\Omega(\log t)$ bits, while $B = o(\log\log n) = o(\log t)$ is again insufficient.

$\Omega(\log\log\log n)$ lower bound. We now turn to the proof of Theorem 1.6, which establishes that the one-round bandwidth complexity of Detect(K_s) under both node insertions and edge insertions is $\Omega(\log\log\log n)$. For clarity, we again focus on the case of membership-detecting triangles.

The key reason this lower bound is exponentially smaller than the previous one is that Detect(K_s) is inherently a much simpler problem than MemDetect(K_s). In particular, if *only edge insertions are allowed*, then triangle detection is solvable with bandwidth $B = O(1)$: When an edge is inserted, its endpoints can simply broadcast a signal to their neighbors, and any node receiving two such signals can locally infer the existence of a triangle.

However, this strategy breaks down when *node insertions* are also permitted. Again, consider a path of the form (a, x_i, b). Now, it is impossible to distinguish whether an edge $\{a, b\}$ has been inserted – completing a triangle – or whether a new node c has been inserted with incident edges $\{a, c\}$ and $\{b, c\}$, where no triangle is created. The ambiguity arises because both scenarios lead to x_i receiving signals from a and b.

To formalize this, we use a construction similar to the one from the previous lower bound, but with a smaller parameter: $t = \log^{0.1}\log n$. We show that distinguishing the two scenarios requires $\Omega(\log\log\log n)$ bits of bandwidth.

Here is a brief sketch of the argument. We label each pair (a, b), for $a, b \in I$, according to the messages transmitted across the edges $\{a, x_i\}$ and $\{b, x_i\}$ in the first $2t$ rounds, under the assumption that the path (a, x_i, b) is formed. Since each message consists of B bits and communication lasts $2t$ rounds, the total number of distinct labels is $s = 2^{O(Bt)}$. Since node insertion is allowed, nodes in I without incident edges are considered as not yet inserted.

A Ramsey-type argument then implies the existence of a subset $\{a, b, c\} \subseteq I$ such that all of (a, b), (a, c), (c, b) receive the same label. This means that, from the perspective of node x_i, the insertion of an edge $e = \{a, b\}$ versus the insertion of a node c with two incident edges $\{a, c\}$ and $\{b, c\}$ becomes indistinguishable.

For the proof to work, the set I must be sufficiently large relative to the number of distinct labels, which is $s = 2^{O(Bt)}$. This requirement is precisely why we set $t = \log^{0.1} \log n$ instead of the larger value $t = \log^{0.1} n$ used in the previous argument, resulting in a weaker lower bound of $\Omega(\log \log \log n)$. The full proof of Theorem 1.6 is more intricate, as it involves analyzing not only the perspective of each x_i, but also the views of the endpoints of the newly inserted edge e. For instance, the actual labeling in the proof requires quantifying over all $i \in [t]$, which increases the number of distinct labels to $2^{O(Bt^2)}$.

3.2 Upper Bounds for Finding Triangles

We now discuss our two triangle-finding algorithms, both of which achieve optimal bandwidth complexity in bounded-degree networks, matching our lower bounds.

Our algorithms build upon the one-round algorithm for MemDetect(K_3) under edge insertions by Bonne and Censor-Hillel [6], which operates with bandwidth $B = O(\sqrt{\Delta \log n})$. Their approach uses two types of messages: one with d bits, and another with $O\left(\frac{\Delta \log n}{d}\right)$ bits. The overall bandwidth complexity is optimized by choosing $d = \sqrt{\Delta \log n}$.

$O(\log \log n)$ upper bound. To prove Theorem 1.4, we extend the algorithm of Bonne and Censor-Hillel to design a one-round algorithm for MemList(K_3) under edge insertions with a bandwidth complexity of $O(\Delta^2 \log \log n)$. In their original algorithm for MemDetect(K_3), each d-bit message is a binary string where the ith bit indicates whether an incident edge was inserted i rounds ago. We observe that this binary string can be compactly represented using $O(\Delta \log d)$ bits: For each recently inserted incident edge, use a number in $[d]$ to indicate its insertion time. Setting $d = \log n$ gives a bandwidth complexity of $O(\Delta \log \log n)$ for MemDetect(K_3).

To handle the more demanding MemList(K_3) problem, we introduce several modifications to the algorithm. Most notably, we must account not only for the edges incident to a node but also for those incident to its neighbors. This additional layer of information increases the size of the message by a factor of Δ, resulting in a total bandwidth complexity of $O(\Delta^2 \log \log n)$.

$O(\log \log \log n)$ upper bound. We now turn to the proof of Theorem 1.7, where we design a one-round algorithm for List(K_3) that handles both edge and node insertions with bandwidth $O(\Delta \log \log \log n)$. The exponential improvement stems from improving the size of the $O\left(\frac{\Delta \log n}{d}\right)$-bit message in the algorithm of Bonne and Censor-Hillel [6] to $O\left(\frac{\Delta \log \log n}{d}\right)$ bits. The purpose of this message is to transmit the list of neighborhood IDs, which contains $O(\Delta \log n)$ bits of information. As the transmission is done in d rounds, the required message size is $O\left(\frac{\Delta \log n}{d}\right)$.

Our key idea lies in a new method for identifying triangles. Recall that a core challenge in our $\Omega(\log \log \log n)$ lower bound proof is to understand the inherent difficulty for a node x, with two non-adjacent neighbors a and b, to distinguish between the insertion of an edge $\{a, b\}$ and the insertion of a node c with two incident edges $\{a, c\}$ and $\{b, c\}$.

We develop a new algorithm to handle this instance. We let x select an index i such that the ith bits of ID(a) and ID(b) differ. Nodes a and b then report the ith bit of the identifier of their new neighbor. If an edge $\{a, b\}$ is inserted, then the reported bits will differ. If a node c, along with two incident edges $\{a, c\}$ and $\{b, c\}$, is inserted, then the bits will match. This comparison allows x to distinguish between the two cases.

Sending an index requires only $O(\log\log n)$ bits, which is exponentially more efficient than sending the full identifier, which requires $O(\log n)$ bits. Consequently, the size of the $O\left(\frac{\Delta \log n}{d}\right)$-bit message in the algorithm of Bonne and Censor-Hillel [6] is reduced to $O\left(\frac{\Delta \log\log n}{d}\right)$ bits. Setting $d = \log\log n$ then yields the improved bandwidth complexity $O(\Delta \log\log\log n)$.

3.3 Membership-Listing

We obtain a *complete characterization* of the bandwidth complexity of MemList(H) for every target subgraph H, every number of rounds r, and every type of topological change: node insertions, node deletions, edge insertions, and edge deletions. The full characterization is summarized in Table 1. In this overview, we omit node insertions and deletions, as they closely mirror the respective cases of edge insertions and deletions.

We begin with the case of *edge insertions*. For $r < r_H$, we have an impossibility result from Theorem 2.11. For $r \geq r_H$, the r-round bandwidth complexity depends on the structure of H:

- If H is a complete multipartite graph that is not a clique, the complexity is $\Theta(n/r)$.
- If H is not a complete multipartite graph, the complexity increases to $\Theta(n^2/r)$.

Upper bounds. The upper bounds follow from the observation that a node v can list all subgraphs H that contain it once it has an accurate view of its r_H-radius neighborhood. A brute-force approach would be to flood the entire graph topology using messages of size $O(n^2)$ after each topological change. Within r_H rounds, this ensures all nodes acquire the required local view. If $r \geq r_H$, this communication can be spread over multiple rounds, reducing the bandwidth requirement to $O(n^2/r)$.

For the special case where H is a complete multipartite graph, we have $r_H = 1$. This allows a more efficient approach: Each node simply broadcasts its list of neighbors as a binary string of length n, leading to a reduced bandwidth complexity of $O(n/r)$.

Lower bounds. We first discuss a general $\Omega(n/r)$ lower bound for any non-clique subgraph H. Let $\{u, v\}$ be a non-edge in H. Consider the graph resulting from replacing u with an independent set U', so each member of U' corresponds to a copy of H. We then construct the graph via edge insertions, ensuring that the edges incident to v are added last, leaving v only $O(r)$ rounds to gather information about the graph. For v to list all copies of H, it must learn the set U', which requires $\Omega(n)$ bits of information, yielding a lower bound of $\Omega(n/r)$.

To strengthen the bound to $\Omega(n^2/r)$ when H is not a complete multipartite graph, we use the fact that such a graph H must contain an edge e and a node v such that neither endpoint of e is adjacent to v. Now we apply a similar construction where e is replaced with a bipartite graph, forcing v to learn the bipartite graph in order to list all copies of H, which requires $\Omega(n^2)$ bits of information.

Edge deletions. We now consider the case of *edge deletions*. As with insertions, we have an impossibility result for $r < r_H$ from Theorem 2.11. For $r \geq r_H$, the r-round bandwidth complexity is $\Theta(n/r)$ whenever H is not a clique.

A key difference between insertions and deletions is that edge insertions can merge disjoint components, potentially introducing many new subgraphs and requiring extensive information dissemination. In contrast, to handle deletions, it suffices to propagate the identifiers of the two endpoints of the deleted edge to a radius of r_H, which requires only $O(\log n)$ bits of information. This yields an upper bound of $O((\log n)/r)$.

To prove a matching lower bound, we reuse the construction from the $\Omega(n/r)$ lower bound. Suppose an edge deletion causes one of the $|U'|$ copies of H to disappear. For a node v to correctly identify *which* copy was affected, it must learn $\Omega(\log |U'|)$ bits. By letting U' contain a polynomial number of nodes, this yields the desired $\Omega((\log n)/r)$ lower bound.

3.4 One-Round Membership-Detection

We now turn to the one-round bandwidth complexity of the MemDetect(H) problem, for which we provide a partial characterization, see Table 2. Compared to MemList(H), establishing lower bounds for MemDetect(H) is more challenging, as it is harder to quantify the minimum information a node must obtain to detect the presence of a subgraph. On the algorithmic side, obtaining optimal upper bounds is also trickier: Since detection does not require listing the subgraph, there is greater flexibility in how the subgraph can be found. This flexibility enables a wider range of algorithmic techniques. In this overview, we focus on two representative results.

Lower bound. We show that the one-round bandwidth complexity of MemDetect(H) under edge insertions is $\Omega(n)$ for any complete multipartite graph H that is neither a star nor a clique. While the corresponding $\Omega(n/r)$ lower bound for MemList(H) appears inherently tied to the listing requirement, we demonstrate that, with suitable modifications, the core idea can be adapted to the detection setting.

Since we cannot require a node v to list all copies of H in the constructed graph, we instead frame the argument in a preprocessing-plus-query model. We first build a graph that initially contains no copy of H, but is structured so that a copy can be formed in many different ways. The goal is to ensure that, in order for v to detect the presence of H following an edge insertion, it must have already learned a significant amount of information about the initial graph during the construction phase.

Specifically, we identify two non-adjacent nodes u and v in H that share a common neighbor w. We construct a graph by removing the edge $\{u, w\}$ from H and replacing node u with an independent set U'. The graph is built via edge insertions, with the edges incident to v and w added at the end, leaving them only $O(1)$ rounds to learn about U'. We then insert a new edge e incident to w, creating two possible scenarios: If the other endpoint of e lies in U', a copy of H is formed; otherwise, it is not. For v to decide correctly, v and w must learn the set U' before the insertion of e, which requires $\Omega(n)$ bits of information.

Upper bound. As in the case of MemList(H), membership-detection becomes significantly easier when the allowed topological change is a deletion rather than an insertion. In particular, we present a one-round $O(1)$-bandwidth algorithm for MemDetect(H) that applies to any complete multipartite graph H, improving upon the $O(\log n)$ bound required for MemList(H) under the same conditions.

For clarity, we describe our algorithm for the special case $H = C_4$, which captures the key ideas behind the more general algorithm that works for an arbitrary complete multipartite graph H. The core observation is that to detect a C_4, it suffices for each node w to maintain, for every pair of its neighbors u and v, the number of common neighbors they share, excluding w. Each such shared neighbor corresponds to a distinct copy of C_4 containing w, u, and v.

Maintaining these counters requires only one-bit messages. When a node detects that one of its neighbors has been deleted, it sends a one-bit signal to all its remaining neighbors. If a node w receives such a signal from two neighbors u and v in the same round, it decrements the counter for the pair $\{u, v\}$ by one.

3.5 One-Round Listing

We study the one-round bandwidth complexity of the $\textsc{List}(H)$ problem for edge deletions and node deletions. See Table 3 for a summary of our results. In this overview, we focus on the case of edge deletions, as the case of node deletions is analogous.

We begin with the simpler cases. When $\widetilde{\text{rad}}(H) = 1$, H is a star, so $\textsc{List}(H)$ is trivially solvable with zero bandwidth by allowing the star center to handle the listing. When $\widetilde{\text{rad}}(H) = 3$, $\textsc{List}(H)$ cannot be solved in one round, since there exists at least one edge e in H whose deletion cannot be communicated to all nodes within one round.

When $\text{rad}(H) = 1$, the graph H has a center node v adjacent to all other nodes, enabling a simple one-round one-bit algorithm: Whenever an edge is deleted, its endpoints send a signal to all their neighbors, allowing the center of H to determine whether H still exists.

We now turn to the remaining nontrivial case, where $\widetilde{\text{rad}}(H) = 2$ and $\text{rad}(H) = 2$. In this setting, we prove a tight $\Theta(\log n)$ bound. The upper bound is achieved by a simple protocol: When an edge $e = \{u, v\}$ is deleted, both u and v broadcast $\text{ID}(u)$ and $\text{ID}(v)$ to all their neighbors. This guarantees that if the deletion eliminates a copy of H, its center, who is responsible for listing the subgraph, can detect the change.

The lower bound is more involved and requires a novel construction. Let $V(H) = \{u_1, u_2, \ldots, u_m\}$. We replace each node u_i in H with an independent set of n nodes $S_i = \{v_{i,1}, \ldots, v_{i,n}\}$. Moreover, we assume, based on the structure of H, that there exists a path of length two from u_1 to u_m via u_{m-1}, but no direct edge between u_1 and u_m.

Using symmetry and the pigeonhole principle, we can assume that node $v_{1,1}$ must list at least $\Omega(n)$ distinct copies of H of the form $\{v_{1,i}, v_{2,1}, v_{3,1}, \ldots, v_{m-1,1}, v_{m,j}\}$ for $i, j \in [n]$. Among these copies of H, there must be at least $\Omega(\sqrt{n})$ copies with distinct i-values, or at least $\Omega(\sqrt{n})$ with distinct j-values.

Assume the former holds, and let I be the set of distinct i values. Now consider deleting the edge $\{v_{1,i^*}, v_{m-1,1}\}$ for some $i^* \in I \setminus \{1\}$. Node $v_{1,1}$ must then stop listing all copies of H that include this edge. Since there is no direct connection between $v_{1,1}$ and v_{1,i^*}, it must receive a message from $v_{m-1,1}$ that uniquely identifies i^* among $\Omega(\sqrt{n})$ candidates, which requires $\Omega(\log n)$ bits. The argument for the latter case (distinct j-values) is similar.

4 Conclusions and Open Problems

In this work, we substantially extend the study of dynamic distributed subgraph finding initiated by Bonne and Censor-Hillel [6] in the *deterministic* setting. We establish *tight* one-round bandwidth bounds for triangle finding in bounded-degree dynamic networks: $\Theta(\log \log n)$ for membership-detection under edge insertions only (Corollary 1.5), and $\Theta(\log \log \log n)$ for detection when both edge and node insertions are allowed (Corollary 1.8). Before our work, no lower bound was known for these two problems. Moreover, we provide a *complete characterization* of the r-round bandwidth complexity of the membership-listing problem across all subgraphs and types of topological changes (Table 1). Despite these advances, many intriguing open problems remain.

Beyond bounded-degree networks While we obtain tight bounds for triangle finding in bounded-degree networks, the current upper and lower bounds remain unmatched for general unbounded-degree networks. Can stronger lower bounds be established for networks with higher degrees?

Specifically, for membership-detection, in Theorem 1.3, we establish a new lower bound of $\Omega(\log \log n)$ for the one-round bandwidth complexity of $\textsc{MemDetect}(K_s)$ under edge

insertions for any $s \geq 3$. While we provide a matching upper bound for MemList(K_3) in *bounded-degree* networks in Theorem 1.4, this lower bound is not yet known to be tight for *unbounded-degree* networks, where the current best upper bound is $O(\log n)$ for $s = 3$ and $O(\sqrt{n})$ for $s \geq 4$ [6]. Closing these gaps remains an intriguing open question.

Randomized algorithms While we focus on deterministic algorithms in this paper, many of our lower bounds extend to randomized algorithms, as shown in the full version [15] of the paper. Yet the role of randomness in reducing bandwidth complexity for dynamic distributed subgraph finding is not well understood: Which problems exhibit an advantage for randomized over deterministic algorithms?

Round-bandwidth tradeoffs While our complete characterization of the membership-listing problem applies to r-round algorithms with an arbitrary round number r, the remainder of our results – and much of the existing literature – primarily focuses on the one-round scenario.

A particularly illustrative case is the membership-listing of cliques: In the one-round setting, the bandwidth complexity is $\Theta(\sqrt{n})$, whereas allowing two rounds reduces the bandwidth complexity to $\Theta(1)$ [6]. This stark contrast shows the potential benefits of additional communication rounds in lowering bandwidth requirements. Exploring how increased round numbers influence bandwidth complexity remains an interesting avenue for future research.

Toward complete characterizations of the remaining problems In this work, we provide partial characterizations for one-round membership-detection and listing. Can these characterizations be completed? What can be said about the detection problem? We discuss several specific open questions in the full version [15] of the paper.

References

1 Amirreza Akbari, Navid Eslami, Henrik Lievonen, Darya Melnyk, Joona Särkijärvi, and Jukka Suomela. Locality in Online, Dynamic, Sequential, and Distributed Graph Algorithms. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.10.

2 Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008. doi:10.1137/07067917X.

3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 815–826, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188922.

4 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 33–42. IEEE, 2019. doi:10.1109/IPDPS.2019.00015.

5 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*, pages 16–24, 2008. doi:10.1145/1839490.1839494.

6 Matthias Bonne and Keren Censor-Hillel. Distributed detection of cliques in dynamic networks. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.132.

7 Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. In Idit Keidar, editor, *Proceedings of the 23rd International Symposium on Distributed Computing (DISC)*, volume 5805 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 2009. doi:10.1007/978-3-642-04355-0_22.

8 Keren Censor-Hillel. Distributed Subgraph Finding: Progress and Challenges. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.3.

9 Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. Tight distributed listing of cliques. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2878–2891, 2021. doi:10.1137/1.9781611976465.171.

10 Keren Censor-Hillel, Neta Dafni, Victor I. Kolobov, Ami Paz, and Gregory Schwartzman. Fast Deterministic Algorithms for Highly-Dynamic Networks. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, volume 184 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2020.28.

11 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019. doi:10.1007/s00446-018-0324-8.

12 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 2020)*, 2020. doi:10.1145/3382734.3405742.

13 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 217–226, 2016. doi:10.1145/2933057.2933083.

14 Keren Censor-Hillel, Dean Leitersdorf, and David Vulakh. Deterministic near-optimal distributed listing of cliques. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 271–280, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538434.

15 Yi-Jun Chang, Lyuting Chen, Yanyu Chen, Gopinath Mishra, and Mingyang Yang. The complexity landscape of dynamic distributed subgraph finding. *arXiv preprint arXiv:2411.11544*, 2024. doi:10.48550/arXiv.2411.11544.

16 Yi-Jun Chang, Shang-En Huang, and Hsin-Hao Su. Deterministic expander routing: Faster and more versatile. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 194–204, 2024. doi:10.1145/3662158.3662797.

17 Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *Journal of the ACM*, 68(3):1–36, 2021. doi:10.1145/3446330.

18 Michael Dinitz, Jeremy T Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of dynamic networks. *Distributed Computing*, 31:273–287, 2018. doi:10.1007/S00446-017-0300-8.

19 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

20 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 2014)*, pages 367–376, 2014. doi:10.1145/2611462.2611493.

21 Chinmoy Dutta, Gopal Pandurangan, Rajmohan Rajaraman, Zhifeng Sun, and Emanuele Viola. On the complexity of information spreading in dynamic networks. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 717–736. SIAM, 2013. doi:10.1137/1.9781611973105.52.

22 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. In *Proceedings of the 33rd*

International Symposium on Distributed Computing (DISC 2019), pages 15:1–15:16, 2019. doi:10.4230/LIPIcs.DISC.2019.15.

23 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017. doi:10.1137/15M1054389.

24 Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 15:1–15:30. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.15.

25 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 153–162, 2018. doi:10.1145/3210377.3210401.

26 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–20, 2019. doi:10.1145/3322811.

27 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *International Symposium on Distributed Computing*, pages 342–356. Springer, 2016. doi:10.1007/978-3-662-53426-7_25.

28 Bernhard Haeupler and David Karger. Faster information dissemination in dynamic networks via network coding. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 381–390, 2011. doi:10.1145/1993806.1993885.

29 Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *The Electronic Journal of Combinatorics*, 24(4):4–21, 2017.

30 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC 2017)*, pages 381–389, 2017. doi:10.1145/3087801.3087811.

31 Irvan Jahja and Haifeng Yu. Sublinear algorithms in T-interval dynamic networks. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 317–327, 2020. doi:10.1145/3350755.3400228.

32 Michael König and Roger Wattenhofer. On local fixing. In Roberto Baldoni, Nicolas Nisse, and Maarten van Steen, editors, *Proceedings of the 17th International Conference on Principles of Distributed Systems (OPODIS), Nice, France*, volume 8304 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2013. doi:10.1007/978-3-319-03850-6_14.

33 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS 2017)*, pages 4:1–4:16, 2017. doi:10.4230/LIPIcs.OPODIS.2017.4.

34 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522, 2010. doi:10.1145/1806689.1806760.

35 Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated consensus in dynamic networks. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 1–10, 2011. doi:10.1145/1993806.1993808.

36 Quanquan C Liu. A note on improved results for one round distributed clique listing. *Information Processing Letters*, 181:106355, 2023. doi:10.1016/J.IPL.2022.106355.

37 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 220–239. SIAM, 2016. doi:10.1137/1.9781611974331.CH17.

38 Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Information and Computation*, 243:263–280, 2015. doi:10.1016/J.IC.2014.12.018.

39 Shay Solomon. Fully dynamic maximal matching in constant update time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334. IEEE, 2016. doi:10.1109/FOCS.2016.43.

40 Haifeng Yu, Yuda Zhao, and Irvan Jahja. The cost of unknown diameter in dynamic networks. *Journal of the ACM (JACM)*, 65(5):1–34, 2018. doi:10.1145/3209665.

A Tables

In the appendix, we provide tables that summarize our results for subgraph finding beyond cliques.

A.1 Membership-Listing

We establish a complete characterization of the bandwidth complexity of r-round MemList(H) for any target subgraph H, for any number of rounds r, under any single type of topological change. We emphasize that, while H is assumed to be a constant-size graph, here we allow r to be a function of n. See Table 1 for a summary of our results. Refer to Definitions 2.10 and 2.12 for the definition of r_H and r'_H.

Table 1 The bandwidth complexity of r-round MemList(H).

	$r \geq r_H$				$r < r_H$
	Complete multipartite graphs ($r_H = 1$)			$r_H \geq 2$	
	Cliques		Others		
	$r = 1$	$r \geq 2$			
Edge insertions	$\Theta(\sqrt{n})$ [6]	$\Theta(1)$ [6]	$\Theta(n/r)$ [New]	$\Theta(n^2/r)$ [New]	Impossible [2.11][2.14]
Node insertions	$\Theta(n/r)$ [6]		$\Theta(n/r)$ [New]	$\Theta(n^2/r)$ [New]	
Edge deletions	$\Theta(1)$ [6]		$\Theta((\log n)/r)$ [New]		

	$r \geq r'_H$		$r < r'_H$
	Cliques ($r'_H = 0$)	Others ($r'_H \geq 1$)	
Node deletions	0 [6]	$\Theta((\log n)/r)$ [New]	Impossible [2.13]

A.2 One-Round Membership-Detection

We investigate the one-round bandwidth complexity of the MemDetect(H) problem under any single type of topological change. A summary of our results is provided in Table 2. For the definitions of rad, diam, and $\widetilde{\text{diam}}$, see Definitions 2.1 and 2.3.

To see that the table for node deletions covers all cases, observe that diam $= 2$ implies rad $\in \{1, 2\}$ because rad $\leq$ diam, and from Observation 2.4 we infer that diam $= 2$ implies $\widetilde{\text{diam}} \in \{2, 3\}$. In the table for the remaining three types of topological changes, refer to the paragraph at the end of Section 2.3 for why complete multipartite graphs characterize one-round solvability.

For node deletions, our $O(1)$-bandwidth algorithm for the case where $\text{diam}(H) = \text{rad}(H) = \widetilde{\text{diam}}(H) = 2$ works for all complete multipartite graphs H. Recall from Observation 2.9 that a connected graph H with at least three nodes is complete multipartite if

Table 2 The bandwidth complexity of one-round MemDetect(H).

	Complete multipartite graphs				Others
	Stars	s-cliques		Others	
		$s = 3$	$s \geq 3$		
Edge insertions	$\Theta(1)$ [New]	$O(\log n)$ [6]	$O(\sqrt{n})$ [6]	$\Omega(\log \log n)$ [1.3]	$\Theta(n)$ [New]	Impossible [2.9]
Node insertions		$\Theta(n)$ [6]			$\Theta(n)$ [New]	[2.11] [2.14]
Edge deletions		$\Theta(1)$ [6]			$O(\log n)$ [New]	

	diam $= 1$	diam $= 2$			diam ≥ 3
		rad $= 1$	rad $= 2$		
			$\widetilde{\text{diam}} = 2$	$\widetilde{\text{diam}} = 3$	
Node deletions	0 [6]	$\Theta(1)$ [New]	$\Theta(1)$ [New]	$O(\log n)$ [New]	Impossible [New]

and only if $\widetilde{\text{diam}}(H) = 2$. For node deletions, the only case where we are unable to obtain a tight bound is when $\text{diam}(H) = \text{rad}(H) = 2$ and $\widetilde{\text{diam}}(H) = 3$. A notable example of such a graph H is the 5-cycle C_5.

A.3 One-Round Listing

We investigate the one-round bandwidth complexity of the problem of List(H) for edge deletions and node deletions. For edge deletions, we obtain a complete characterization. For node deletions, we obtain an almost complete characterization, except for the case where $\text{rad}(H) = \text{diam}(H) = 2$. See Table 3 for a summary of our results. Refer to Definitions 2.1 and 2.3 for the definition of rad, $\widetilde{\text{rad}}$, and diam.

Table 3 The bandwidth complexity of one-round List(H).

	$\widetilde{\text{rad}} = 1$	$\widetilde{\text{rad}} = 2$		$\widetilde{\text{rad}} \geq 3$
		rad $= 1$	rad $= 2$	
Edge deletions	0 [New]	$\Theta(1)$ [New]	$\Theta(\log n)$ [New]	Impossible [New]

	rad $= 1$	rad $= 2$		rad ≥ 3
		diam $= 2$	diam ≥ 3	
Node deletions	0 [New]	$O(\log n)$ [New]	$\Theta(\log n)$ [New]	Impossible [New]

Boosting Payment Channel Network Liquidity with Topology Optimization and Transaction Selection

Krishnendu Chatterjee
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Jan Matyáš Křišťan
Faculty of Information Technology, Czech Technical University, Prague, Czech Republic

Stefan Schmid
TU Berlin, Germany
Weizenbaum Institute, Berlin, Germany

Jakub Svoboda
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Michelle Yeo[1]
National University of Singapore, Singapore

Abstract

Payment channel networks (PCNs) are a promising technology that alleviates blockchain scalability by shifting the transaction load from the blockchain to the PCN. Nevertheless, the network topology has to be carefully designed to maximise the transaction throughput in PCNs. Additionally, users in PCNs also have to make optimal decisions on which transactions to forward and which to reject to prolong the lifetime of their channels. In this work, we consider an input sequence of transactions over p parties. Each transaction consists of a transaction size, source, and target, and can be either accepted or rejected (entailing a cost). The goal is to design a PCN topology among the p cooperating parties, along with the channel capacities, and then output a decision for each transaction in the sequence to minimise the cost of creating and augmenting channels, as well as the cost of rejecting transactions. Our main contribution is an $\mathcal{O}(p)$ approximation algorithm for the problem with p parties. We further show that with some assumptions on the distribution of transactions, we can reduce the approximation ratio to $\mathcal{O}(\sqrt{p})$. We complement our theoretical analysis with an empirical study of our assumptions and approach in the context of the Lightning Network.

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Blockchains, Cryptocurrencies, Payment Channel Networks, Throughput, Optimisation, Graph Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.23

Related Version *Full Version*: `https://eprint.iacr.org/2025/1484.pdf`

Funding *Krishnendu Chatterjee*: European Research Council CoG 863818 (ForM-SMArt) and Austrian Science Fund 10.55776/COE12.
Jan Matyáš Křišťan: Czech Science Foundation Grant no. 24-12046S.
Stefan Schmid: German Research Foundation (DFG) project ReNO (SPP 2378) from 2023-2027.
Jakub Svoboda: European Research Council CoG 863818 (ForM-SMArt) and Austrian Science Fund 10.55776/COE12.
Michelle Yeo: MOE-T2EP20122-0014 (Data-Driven Distributed Algorithms).

[1] Corresponding author

© Krishnendu Chatterjee, Jan Matyáš Křišťan, Stefan Schmid, Jakub Svoboda, and Michelle Yeo;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Blockchain scalability is one of the key bottlenecks in the mainstream adoption of cryptocurrencies [9, 12, 24]. For instance, Bitcoin can only process an average of 7 transactions per second which is paltry compared to Visa's 47,000 [26]. This makes it impractical as of yet for the widespread usage of cryptocurrencies in everyday situations.

Some promising solutions to blockchain scalability are layer 2 solutions like payment channel networks (PCNs) [13, 14, 25, 26], which allow users to bypass the blockchain and transact directly with each other. To do so, two users have to first open a payment channel between themselves in an initial funding transaction on the blockchain, thereby locking some funds only to be used in the channel. Thereafter, the two users transact with each other simply by updating the distribution of funds on each end of the channel to reflect the payment amount. As compared to the tedious process of waiting for consensus on the blockchain, these payments can be finalised almost instantaneously since they only involve exchanging signatures between the two parties. As such, with only a constant number of blockchain transactions, any two users can make an unbounded number of off-chain transactions in the payment channel, consequently increasing the overall transaction throughput of the blockchain.

A PCN is a network of payment channels and users, where two users that are not directly connected to each other with a payment channel can still transact with each other in a multi-hop fashion over a path of payment channels. To incentivize intermediary nodes on payment paths to forward payments, PCNs allow these intermediary nodes to charge a small transaction forwarding fee that is typically linear in the transaction amount. Two of the most notable examples of PCNs are Bitcoin's Lightning Network [26] and Ethereum's Raiden [2].

Apart from joining a PCN in order to transact efficiently with other users, a secondary consideration for users in PCNs is to whom they should establish channels and at which capacities, as well as which transactions then to select for forwarding. Aside from the opportunity cost of rejecting to forward transactions and thus missing out on the revenue from transaction fees, users in PCNs also incur a cost whenever channels are created (the on-chain transaction fee) as well as when additional capacity is injected into the channel. As such, although it is tempting to accept to forward all transactions and thus profit from the transaction fees, doing so would quickly deplete the nodes' capacity on their incident channels. A crucial optimisation problem that intermediary nodes in transaction paths would therefore have to consider is deciding which transactions to accept and which to reject, given a minimal injection of capacity into their channel.

In this work, we study the problem of both optimal topology design and transaction selection over a PCN. The input to the problem is a sequence of transactions and a set of p coordinating parties. Each transaction in the sequence is of a certain amount, and is associated with a source and target from the set of parties. The problem then proceeds in two stages: in the first stage, the goal is for the p parties to decide jointly on a PCN topology over themselves. Each created channel in this stage incurs both a fixed channel creation cost as well as a capacity cost that depends on the amount of funds injected into the channel. The second stage involves processing the transactions in the transaction sequence, either accepting or rejecting them (and in so doing incurring a rejection fee) along the created paths in the network. The goal is to create a network and inject enough capacity into each channel such as to maximise the number of transactions accepted while incurring the least amount of cost (see the cost model in Section 2 for more details).

1.1 Related work

To the best of our knowledge, our work is the first to consider the problem of designing an optimal PCN topology for optimising the problem of transaction selection in PCNs. Our work is closely related to the work of [28] which considers a similar problem over a single payment channel and shows that the problem of deciding which transactions to accept or reject over a single channel is already NP-hard. The main open question of [28] is how to extend their proposed algorithm from a single channel to an entire network. Our work addresses this problem by proposing a solution for network design as well as algorithms for optimally accepting or rejecting transactions along paths in a network. Our work is also related to the works of [4, 7] which consider a similar problem of transaction selection but in the online setting and also limited to a single channel. Additionally, the works of [16, 17] also study minimal-cost PCN topologies, but do not consider the complementary problem of transaction selection.

Another related line of research is network creation games on cryptocurrency networks [5, 6, 15]. These works study how a node should connect to an existing cryptocurrency network in a stable and optimal way, optimising the utility of the single node in a unilateral fashion. In contrast, our work proposes a method of creating a network among any number of nodes such as to *jointly minimise* the cost (over all nodes) of network creation and processing transactions.

A crucial assumption in our model is coordination between the interested parties in creating the network during the network creation stage. We stress that this is not an impractical assumption and several works have made similar assumptions [5, 6, 7] where optimal behaviour is analysed in the setting where some or all involved parties cooperate and create certain network topologies together.

Further upfield but also related are works analysing the consensus number of a cryptocurrency [20, 22]. The transactions studied in our work, and in general the sequential nature of layer 2 transactions on PCNs, have consensus number 1, implying that expensive consensus protocols are unnecessary in such settings. Additionally, our work is also related to the classic works on theory of contention management in transactional memory [3, 18, 19].

More generally, our work is related to optimising flows and throughput in typical capacitated communication networks [10, 27]. However, we stress a crucial difference between our problem and problem over traditional communication networks: in traditional communication networks, the capacity is usually independent in the two directions of the channel [21]. In our case, the amount of transactions u sends to v in a channel (u, v) directly affects v's capability to send transactions, as each transaction u sends to v increases v's capacity on (u, v).

1.2 Main challenges and our contributions

We generalise the model of processing transactions over a single channel proposed in [28] to an entire network. The first challenge in shifting from optimising processing transactions from a single channel to a network is deciding on the optimal network topology among the cooperating parties. Here, we show in Theorem 2 in Section 3.2 that the star graph of size p is a 2-approximation of the problem of creating an optimal PCN over p parties.

Once the optimal topology is decided, the next challenge would be to decide on which transactions to accept to forward, given that transactions have to be accepted along the entire payment path. In this setting, we leverage the star topology to generalise the existing single channel algorithm in [28] to get an algorithm for transaction selection optimisation in the star. Our main algorithm in this setting is described in Section 4 and we prove that it is an $\mathcal{O}(p)$ approximation of the weighted transaction selection problem in general graphs.

We describe another algorithm in Section 5 that has an improved approximation ratio of $\mathcal{O}(\sqrt{p})$ (see Theorem 10) for the transaction selection problem in general graphs. To achieve this approximation ratio, we adopt an additional realistic and reasonable assumption on the distribution of transactions in the transaction sequence. Namely, we assume that transactions are clustered, following a stochastic block model.

Finally, we augment our theoretical analysis with a case study of the Lightning Network in Section 6. Using simulations, we find that the network statistics correspond to and thus justify our transaction distribution assumptions.

2 Model

Payment channels. A unique aspect of our problem is that we optimise the selection of weighted transactions over payment channels. Payment channels can be formally modeled as edges that satisfy the following three properties:

1. The capacity or weight of a channel (u, v) is the sum of the initial capacities b_u and b_v injected into the channel by users (or nodes) u and v. The capacities b_u and b_v correspond to the initial amount of funds that users u and v choose to inject into the channel during channel creation.
2. Given a capacitated channel (u, v) with weight w, the weight or capacity can be arbitrarily split between both ends of the channel depending on the number and weight of transactions processed by u and v. That is, b_u and b_v can be arbitrary as long as $b_u + b_v = w$ and $b_u, b_v \geq 0$. For instance, given a payment channel (u, v) with an initial capacity split of b_u and b_v, if u sends a payment of amount x to v, the capacity on u's end of the channel drops to $b_u - x$ and the capacity of v's end increases to $b_v + x$ after processing the transaction. See Figure 1 for more examples of how the capacities on each end of a payment channel can vary in the course of processing transactions.
3. The total capacity of a payment channel is invariant throughout the lifetime of the channel. That is, it is impossible for nodes to add to or remove any part of the capacity in the channel. In particular, if a node is incident to more than one channel in the network, the node cannot transfer part of its capacity in one channel to "top up" the capacity in the other one.

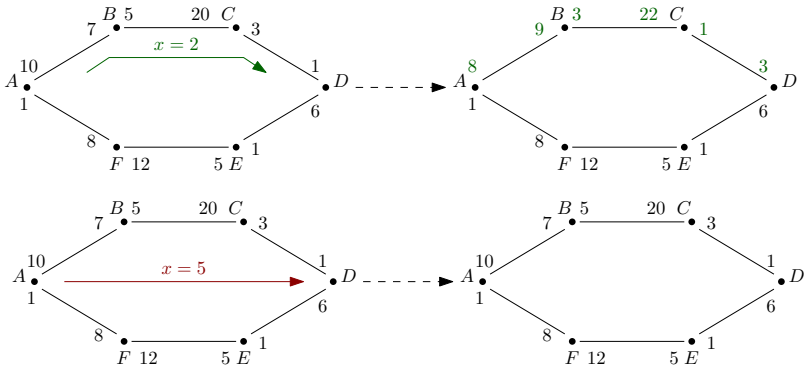


Figure 1 The diagram on the top shows the outcome of processing a transaction of size 2 going from A to D along the path (A, B, C, D) in the graph. The diagram on the bottom shows the outcome of processing a transaction of size 5 from A to D. The transaction has to be rejected as the size of the transaction is larger than the sum of the capacities of C in the channel (C, D) and either A or E in the channel (A, F) or (E, D) respectively.

Payment channel network. We model a PCN as a weighted, undirected graph $G = (V, E)$ where the vertices or nodes of G are users in the PCN and the edges of G are payment channels between nodes. We denote the capacity of a channel $(u, v) \in E$ as $w_{u,v}$, and the capacity of nodes u and v on the channel (u, v) as $b_{u,v}$ and $b_{v,u}$ respectively.

Transaction sequence. Let V denote a set of nodes (parties). We denote a finite ordered sequence of transactions over V by $X_n = ((s_1, t_1, x_1), \ldots, (s_n, t_n, x_n))$, where $s_i, t_i \in V$ represents the sender and target of the ith transaction respectively and $x_i \in \mathbb{R}^+$ represents the weight or size of the ith transaction.

Processing transactions in PCNs. Let $G = (V, E)$ be a connected PCN with some capacity over all of its channels. Let X_n be a sequence of transactions over V. Consider the ith element of the sequence (s_i, t_i, x_i) and suppose the transaction travels over some path $\pi_i = (s_i, \ldots, t_i)$ from the ith sender to the ith target. Let us call the first node along each channel in the path π_i the *forwarding node*. Each forwarding node in the path π_i can choose to do the following to the transaction:

- **Accept transaction.** For a forwarding node u in a channel $(u, v) \in \pi_i$, node u can accept to forward the transaction if their capacity in (u, v) is at least the size of the transaction.
- **Reject transaction.** Node u can also reject the transaction. This could happen if u's capacity is insufficient, or if accepting the transaction would incur a larger cost in the future. For a transaction of weight x, the cost of rejecting the transaction is $f \cdot x + m$ where $f, m \in \mathbb{R}^+$. Apart from incurring some cost, rejecting a transaction does not alter the capacity distribution on both ends of the channel (see the diagram on the bottom of Figure 1 for an example of rejecting a transaction).

We note that node v does not need to take any action nor incur any cost when transactions are going in the direction of u to v along a channel (u, v). Finally, we stress that decisions on each channel in a transaction path need to be the same.

Problem definition. Suppose we have p coordinating parties that want to come together to create a PCN that allows the processing of transactions between themselves. The input to the problem is a transaction sequence over the p coordinating parties. The problem then proceeds in 2 stages. The first stage of the problem is the *network creation stage*, where the p parties have to firstly decide on a PCN topology over themselves. Once the PCN has been created, the second stage of the problem is *transaction selection*, where the network topology created in stage 1 determines the path the transactions travel over. In this stage of the problem, the parties need to decide whether to accept or reject each transaction in the sequence.

Hardness. We know from [28] that the transaction selection problem over a single channel is NP-hard. Our problem includes $p > 2$ parties and we also need to find the optimal network topology over these p parties (for two parties, it is only a channel). This means that our problem is at least as hard as the problem on a single channel, and hence NP-hard as well.

Objective and costs. We consider three types of costs faced by the p parties:

1. **Channel creation costs.** We assume creating each channel incurs a fixed auxiliary cost of $k > 0$.

2. **Capacity costs.** For each channel (u, v) created between parties u and v, the capacity cost of (u, v) is the sum of the capacities $b_{u,v}, b_{v,u}$ injected by u, v into the channel.
3. **Rejection costs.** These are costs incurred during transaction selection when a party rejects a transaction.

The *total cost* incurred by the parties is the sum (over all channels in the PCN and transaction decisions) of channel creation, capacity, and rejection costs. The objective of the p parties is to create a PCN over themselves and output the decisions for each transaction in the sequence such that the total cost is minimised.

3 Accepting all transactions

We begin our investigation of transaction selection over general graphs by first considering the problem a restricted setting where all transactions in the input transaction sequence need to be accepted. In this setting, the total cost consists only of channel creation and capacity costs. We also assume here that we know the optimal PCN topology among the p parties, which we denote by $G = (V, E)$.

3.1 Linear program for a set of channels

We present a linear program (LP) that, for a given set of channels, computes the optimal capacity that the parties need to lock into the channels of the network in order to accept all transactions in the sequence. Indeed, if the channel creation cost k is small or if we want to know a lower bound for our problem, we can assume the optimal graph G is complete. For a node $v \in G$, let $Ne(v)$ denote its neighbours. Let S_v denote the *total balance* of node v in the graph, which is simply the sum of its balances on all its incident channels. For a channel $(u, v) \in E$, we denote the balance of node v (resp. u) after processing the ith transaction in the sequence by $b^i_{v,u}$ (resp. $b^i_{u,v}$), and we also denote the total balance of v after processing the transaction by S^i_v. Let $AC(\mathcal{S}, G)$ denote the optimal capacity cost for the graph G where every transaction in the transaction sequence $\mathcal{S} := ((\pi_1, x_1), \ldots, (\pi_n, x_n))$ over G is accepted. That is, the total capacity injected into the channels of the graph G.

Before describing the LP, we define a *difference function* $\delta_v(s, t)$ between a source s and target t in a transaction path:

$$\delta_v(s,t) = \begin{cases} 1 & v = t \text{ and } v \neq s \\ -1 & v = s \text{ and } v \neq t \\ 0 & \text{otherwise} \end{cases}$$

This function will be used in the LP to maintain the invariant that processing a transaction can only affect the total balances of the source and target nodes in the transaction path, but the total balance of every other node in the graph must not change.

Given G and a transaction sequence $\mathcal{S}$ over G, we construct the following LP.

$$\begin{aligned} \text{minimise} \quad & \sum_{(v,u)\in E} w_{v,u} \\ \forall v, u, i: \quad & w_{v,u} = b^i_{v,u} + b^i_{u,v} \\ \forall v, i: \quad & S^i_v = \sum_{u \in Ne(v)} b_{v,u} \\ \forall v, i, (s_i, t_i, x_i): \quad & S^i_v - \delta_v(s_i, t_i) x_i = S^{i-1}_v \end{aligned}$$

For the sequence of transactions $\mathcal{S}$ and graph G, we denote the solution of the LP with the respective parameters by $LP(\mathcal{S}, G)$. The following lemma (with proof in Section A.1) proves that the solution of the LP is a lower bound on the capacity cost of the problem in the setting where all transactions are to be accepted.

▶ **Lemma 1.** *Given the sequence of transactions $\mathcal{S}$, we have $LP(\mathcal{S}, K_p) \leq AC(\mathcal{S}, G)$.*

3.2 Star as a 2-approximation

Although the capacity cost for the setting where all transactions are accepted is lowest in the complete graph K_p, the complete graph has $\Theta(p^2)$ edges. Thus, depending on the auxiliary channel creation cost k, the cost of creating K_p as well as injecting sufficient capacity into all channels of K_p might be large.

As a useful heuristic, we observe from the proof of Lemma 1 that the capacity cost is related to the diameter of the graph: the smaller the diameter, the lower the cost. Thus, a useful class of graphs to look at for the network creation stage of the problem are graphs that have both a small diameter and a small number of channels. An example of a graph that simultaneously satisfies both of these requirements is a star. In a star, one central node is connected by a channel to all other nodes. Thus, the star has a small diameter of 2, and $p-1$ channels, which is minimal for a connected graph over p nodes. We denote the star on p nodes as $K_{1,p-1}$.

We first observe that for the star, it is easy to compute a lower bound on the amount of capacity required in the setting where all transactions are accepted. This is due to the fact that between any two nodes u, v in a star, there is only one unique path channeling u and v, rendering transactions unsplittable. The capacity on the channel between any node v and the centre of the star c, $w_{v,c}$, has to therefore be larger than $\max_{j,k \leq n} \quad |\sum_{s_i,t_i,x_i | k \leq i \leq j} \delta_v(s_i, t_i) \cdot x_i|$, as this quantity captures the largest difference in the balance between both users of the channel.

We now show in Theorem 2 that we can upper bound the optimal capacity cost on the star. The proof of Theorem 2 is in Section A.2.

▶ **Theorem 2.** *For every graph G over p nodes, and any graph G' with diameter d, the following inequality holds*

$$AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)\,.$$

▶ **Corollary 3.** *For every graph G over p nodes, and graph $K_{1,p-1}$, the following inequality holds*

$$AC(\mathcal{S}, K_{1,p-1}) \leq 2 \cdot AC(\mathcal{S}, G)\,.$$

With these insights, we restrict the network created among the p parties during the network creation stage of the weighted transaction problem to a star in Section 4, and a star of stars (of diameter 4, see Figure 3 for an example) in Section 5.

4 A general setting approximation algorithm

We now present an approximation algorithm for the general setting where transactions can be either accepted or rejected. We first reiterate that from Theorem 2 we can restrict ourselves to the star graph at the network creation stage of the problem, and in doing so incur twice as much capacity cost as the optimal capacity cost. Given that the channel creation cost for

the star topology is fixed at $k(p-1)$, the focus of our algorithm in this section is to come up with capacities that should be injected on both ends of each channel as well as a sequence of transaction decisions so as to minimise the capacity and rejection costs.

4.1 LP for a star

We describe a LP that minimises the capacity cost as well as rejection cost over transactions going between any two leaf nodes in the star. This LP is a relaxation of the general problem, where we allow some transactions to be accepted fractionally. That is, for a transaction of size x_i, the LP does not have to accept the full transaction amount, but can accept some portion y_i such that $0 \leq y_i \leq x_i$. Let us call transactions for which $y_i = x_i$ in the LP solution *fully accepted* transactions.

Let c denote the central node of the star. For a leaf node v and $i \in \mathbb{N}$, we use $C_{v,i}$ and $C'_{v,i}$ to denote the capacity at the channel (v, c) after processing the i-th transaction on the side of v and c respectively. Let C_v denote the total capacity of the channel (v, c). We can now formulate the LP as follows:

$$\begin{aligned}
\text{minimise } & \sum_u C_u + \sum_i f(x_i - y_i) + \frac{m(x_i - y_i)}{x_i} && (1)\\
\forall i : & 0 \leq y_i \leq x_i \\
\forall (v, u, x_i) : & C_v, C_u \geq x_i \\
& C_{v,i} = C_{v,i-1} - y_i \\
& C'_{v,i} = C'_{v,i-1} + y_i \\
& C'_{u,i} = C'_{u,i-1} - y_i \\
& C_{u,i} = C_{u,i-1} + y_i
\end{aligned}$$

▶ **Lemma 4.** *The total cost of the star LP solution is a lower bound on the total cost of the optimal solution.*

Proof. The proof follows from the observation that the optimal integer solution is also a feasible solution for the LP. ◀

▶ Remark 5. The solution of the LP assumes that the capacity cost and the transaction rejection cost are equally large, i.e., we cannot aggressively optimise for one objective while neglecting the other. An implication of this which motivates the design of our algorithms in the following sections is that our algorithms should infrequently reject fully accepted transactions, as these are transactions that do not incur any rejection cost and also change the distribution of the capacity along channels in a more optimal way.

4.2 An $\mathcal{O}(p)$ approximation algorithm

Algorithm for a single channel. We summarize an algorithm from [28] that solves the problem for a single channel, which we will call the *channel algorithm*. The channel algorithm first approximates the capacity to be injected into the channel. Next, the channel algorithm solves a LP (similar to Equation (1)). Given the solution of the LP, we denote by C_u and C_v the minimal capacity needed on each end of a channel (u, v) to mimic the (fractional) transport of transactions as dictated by the LP. In addition to C_u and C_v, the channel algorithm allocates additional capacities on each end of the channel, which we call *reserves*, and denote them by R_u and R_v. These reserves are mainly used to ensure that all transactions that are fully accepted by the LP are also accepted by the channel algorithm (a consequence of Remark 5).

Extending the channel algorithm to a star. We describe our algorithm that extends the channel algorithm over 2 nodes to a star $K_{1,p-1}$ with central node c in Algorithm 1. The key idea behind our algorithm is to first use the LP as described in Equation (1) to fix initial capacities on each end of the channel for every channel in the network (for instance the capacity C_v for v in (v, c) in Figure 2). We also provide each leaf with $p-2$ additional reserves (to process transactions for each other leaf). For each leaf pair u and v, let M denote the smallest change in the capacity as specified by the output of the LP after all interactions between u and v. The additional reserve R_u^v of u for processing transactions of v would then be of size $\frac{\sqrt{3}}{2}M$ (similarly for v). We leave details of Algorithm 1 to Section B.1.

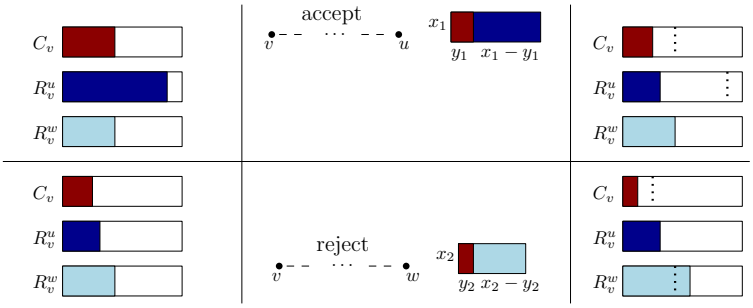


Figure 2 Movement of capacities and reserves from the perspective of node v in a star with centre node c and 2 other leaf nodes u and w. C_v represents the capacity as specified by the output of the LP. R_v^u and R_v^w denote the reserves stored by v to process transactions from u and w respectively. The rightmost column represents the capacities and reserves of v after processing the transaction.

The next theorem shows that Algorithm 1 is an $\mathcal{O}(p)$ approximation of the optimal cost on a star. The proof of Theorem 6 is in Section A.3

▶ **Theorem 6.** *Given the transaction sequence X_t, let $\mathcal{C}$ be the optimal capacity cost and $\mathcal{R}$ the optimal rejection cost on a star. Then Algorithm 1 run for every pair of nodes incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$.*

In the next section, we show that if we impose some reasonable assumptions on the distribution of the input transaction sequence in our model, we can design an algorithm with a significantly lower approximation ratio of $\mathcal{O}(\sqrt{p})$.

5 An improved algorithm using the stochastic block model

Stochastic block model. In reality, users typically interact with each other in a structured manner, resulting in spatial and temporal locality. For example, if two nodes that previously transacted with each other are more likely to transact again in the near future. Our model for how nodes interact and send transactions to each other is the stochastic block model [23], in particular, the planted partition model [11]. The stochastic block model is a generative model to create random graphs where nodes are partitioned into clusters (blocks). See Section C.1 for more details on the stochastic block model.

Clustering. We now define the way nodes are partitioned into clusters given an input sequence of transactions. Given an input transaction sequence over a network, we say that the nodes are (m, k, t)-*clustered* if there exists a partition of nodes to m clusters $C_1, C_2, \ldots, C_m$ where each cluster has size at most k and the clusters satisfy the following *clustering conditions*:

Algorithm 1 Modified channel algorithm.

Input: nodes u and v, transaction sequence X_t, solution of LP indexed by i:C_u, C'_u, C_v, C'_v.
Output: decisions to accept or reject for transactions between u and v

$M \leftarrow \min(C_{u,0} + C'_{u,0}, C_{v,0} + C'_{v,0})$
$R_u = \frac{\sqrt{3}}{2}M$, $R_v = \frac{\sqrt{3}}{2}M$
$H \leftarrow$ CreateHeap()
for $i \in [t]$ **do**
  if x_i is transaction from u to v **then**
    if $R_u - (x_i - y_i) \geq \frac{\sqrt{3}-1}{2}M$ **then**
      $x_i \leftarrow$ **Accept**
      $R_u, R_v = R_u - (x_i - y_i), R_v + (x_i - y_i)$
    else if $\frac{y_i}{x_i} < \frac{\sqrt{3}}{\sqrt{3}+1}$ **then**
      $x_i \leftarrow$ **Reject**
      $R_u, R_v = R_u + y_i, R_v - y_i$
    else
      $x_i \leftarrow$ **Accept**
      $R_u, R_v = R_u - (x_i - y_i), R_v + (x_i - y_i)$
      AddToHeap(H, x_i)
      if $R_u < 0$ **then**
        while $R_u < \frac{\sqrt{3}-1}{2}M$ **do**
          $x_j \leftarrow$ PopHeap(H)
          $x_j \leftarrow$ **Reject**
          $R_u, R_v = R_u + x_j, R_v - x_j$
  if $R_u \geq \frac{\sqrt{3}-1}{2}M$ **then**
    EmptyHeap(H)

1. The volume (sum of sizes of transactions) of transactions inside the cluster is t-times the volume of transactions that go outside the cluster. That is,

$$\sum_{(s_j,t_j,x_j)|s_j,t_j \in C_i} x_j \geq t \cdot \sum_{\substack{(s_j,t_j,x_j)|s_j \in C_i \text{ and } t_j \notin C_i \\ \text{or } s_j \notin C_i \text{ and } t_j \in C_i}} x_j$$

2. For every $S \subset C_i$ such that $|S| \leq \frac{C_i}{2}$, the volume of transactions between S and $C_i \setminus S$ is at least $\frac{|S|}{4|C_i|}$ the volume in the cluster C_i. That is,

$$\frac{|S|}{4|C_i|} \sum_{x_j \text{ inside } C_i} x_j \leq \sum_{x_j \text{ between } C_i \setminus S \text{ and } S} x_j$$

3. For every $S \subset C_i$, the ratio of volumes of transactions from S and $C_i \setminus S$ is at least $\frac{1}{3}$. That is,

$$\frac{\sum_{x_j : C_i \setminus S \to S} x_j}{\sum_{x_j : S \to C_i \setminus S} x_j} \geq \frac{1}{3}.$$

These conditions need to be satisfied for every cluster C_i and any contiguous subsequence of transactions involving C_i (i.e., source or target in C_i) with volume at least $C_i \cdot x_{max}$, where x_{max} is the weight of the biggest transaction in the input transaction sequence. In the rest of this section, we assume that given an input sequence of transactions, the nodes can be

$(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered. Looking ahead, this allows us to design an algorithm that gives us an $\mathcal{O}(\sqrt{p})$ approximation of the optimal cost. In the general case, for an (m, k, t)-clustering, for any $t \geq 24\sqrt{p}$, we have an $\mathcal{O}(m + k)$ approximation (by a similar analysis).

Double star. We define a double star as a graph of diameter $d = 4$ with one central node, some middle nodes, and some leaf nodes. The central and middle nodes create a star and the leaf nodes are connected to only one middle node, forming a cluster (see Figure 3 for an example with 6 clusters). In each cluster, the leaf nodes are connected to the middle node in a star topology. In our setting, given our aforementioned assumption on $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustering, parties create a double star network topology during the network creation stage, where each cluster represents a group of nodes of size $\sqrt{p}$ that interact frequently with each other.

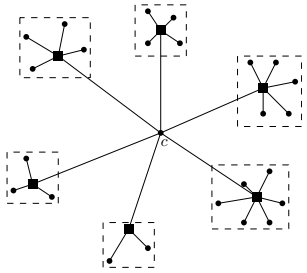


Figure 3 Double star graph with 6 clusters in the dashed boxes. The node labelled c is the centre node, and the middle nodes are the square nodes.

5.1 An $\mathcal{O}(\sqrt{p})$ approximation algorithm

Main ideas. We describe our algorithm that gives an $\mathcal{O}(\sqrt{p})$ approximation of the optimal cost of the problem in Algorithm 2. On a high level, Algorithm 2 consists of two main ideas. The first idea is to again formulate and solve an LP on a double star to get the amount of capacities nodes should inject on their ends of their incident links. This LP is similar to the LP for a star described in Equation (1) and we detail it in Section C.2. From Theorem 2, we know that the capacity cost of the double star is at worst a 4-approximation of the optimal capacity cost.

The second idea is to split the processing of transactions into processing transactions *between* clusters and processing transactions *within* clusters. Here, we use the specific double star topology to ensure that we can apply Algorithm 1 (for processing transactions on a single star) to process transactions between clusters as well as within each cluster. Finally, we use our clustering assumptions to allocate enough reserve capacity for each cluster to ensure that we can always shift the reserve capacities within a cluster from nodes with high to low reserves. This ensures that each node always has sufficient reserve capacity to align the decisions on their transactions (whether to accept or reject) as closely as possible to the ideal decisions as output from the linear program.

Processing transactions between clusters. After running the linear program for the double star, let us rewrite each transaction in the sequence as (s, t, x_i, y_i), where s is the source node, t is the target node, x_i is the weight of the transaction, and y_i denotes the fractional amount of the transaction accepted by the solution of the linear program. For processing transactions between clusters, we look at the star created by the centre and middle nodes of the double

Algorithm 2 Approximation algorithm for clustered nodes.

Input: Sequence of transactions X_t with LP solution, clustering $\mathcal{C} = \{C_1, C_2, \dots\}$
Output: Decisions to accept or reject transactions
1: $X_i \leftarrow \textsc{BetweenClusters}(X_t, \mathcal{C})$
2: $S_i \leftarrow \textsc{Solve}(X_i)$
3: **for** $C \in \mathcal{C}$ **do**
4: $\quad X_C \leftarrow \textsc{BalanceInCluster}(X_t, S_i, C)$
5: $\quad S_C \leftarrow \textsc{Solve}(X_C)$
6: **return** $S_i \cup S_b \cup (\bigcup_{C \in \mathcal{C}} X_C)$

star and treat these transactions as going only between these middle nodes. In Algorithm 2, this is handled by the function BetweenClusters which selects only transactions that go between clusters and treats clusters as a single node. As this becomes processing transactions on a star, this allows us to use Algorithm 1 to process transactions between different clusters. In Algorithm 2, this is handled by the function $\textsc{Solve}(X)$ in line 2 in Algorithm 2 which outputs the decisions using Algorithm 1 for every pair of nodes. Recall that Algorithm 1 also gives us an upper bound on the *reserve capacity* requirement for each cluster (middle node): for each middle node, we require reserves of size $\mathcal{O}(\sqrt{p}M)$, where M is the maximum of the capacity of the channel between the middle node and the centre of the double star c and the capacities of the channels between the middle node and the leaf nodes in its cluster. We call these reserves *cluster reserves*.

Processing transactions within clusters. Before processing transactions that go between nodes in a cluster, it is imperative that the cluster reserves are first shifted around within the cluster to ensure each leaf node in the cluster has enough reserves to process their between-cluster transactions (line 2 in Algorithm 2). This is handled by the function BalanceInCluster (described in Algorithm 3 in Section C.3), which modifies the solution of the double star LP such that cluster reserves are transferred from well-funded to depleted nodes. Finally, after balancing the reserve capacities, Algorithm 2 treats each cluster as an independent star graph and uses Algorithm 1 to process transactions within each cluster (function Solve in line 5). Using Algorithm 1 imposes a reserve capacity requirement of $\mathcal{O}(\sqrt{p}M')$ times the capacity for every leaf node, where M' is the capacity of the channel between the leaf node and the middle node.

Properties of balancing reserve capacities. We now prove several important properties of transporting reserve capacities within clusters as specified by BalanceInClusters, ensuring the correctness of BalanceInClusters. First, Lemma 7 states that we can modify the solution of the linear program (i.e., the fractional amount of transactions accepted by the linear program) to move the reserves capacities from the source s to target t, see Figure 4. From the statement of Lemma 7, we note that the only transactions that do not allow moving reserves are transactions that are fully accepted ($y_i = x_i$) or rejected ($y_i = 0$). Then, we quantify how much capacity we can move from one set of nodes to another in Lemma 8. Lastly, we show in Lemma 9 that we can always transport some reserves between any two sets of nodes. We defer the proofs of these lemmas to Sections A.4–A.6.

▶ **Lemma 7.** *If there is a transaction (s, t, x_i, y_i), we can change it to*
- *$(s, t, x_i, y_i + \alpha)$ to simulate moving α reserves from s to t, and*
- *$(s, t, x_i, y_i - \alpha)$ to simulate moving α reserves from t to s.*

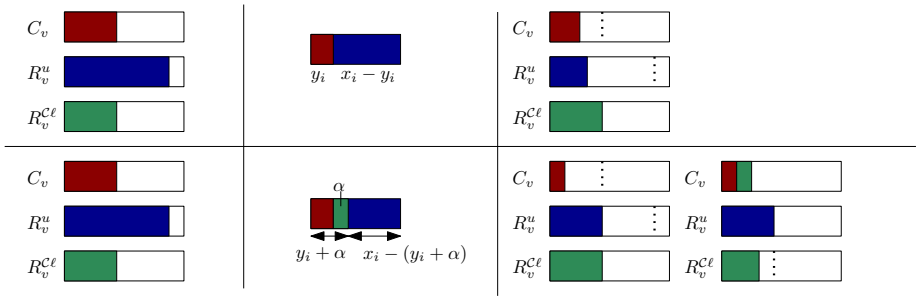


Figure 4 Movement of capacities, reserves and cluster reserves when accepting a transaction x_i from v to u where $v, u \in C_i$ for some cluster C_i. The case on the top represents the typical movement of capacities and reserves when accepting x_i. The case on the bottom represents the treatment of the reserves (and cluster reserves) when the change in cluster reserves ($R_v^{C\ell}$) is involved. The highlighted reserves are on v's side of the channel between v and the middle node. Similar changes are made on u's side of the channel between u and the middle node.

▶ **Lemma 8.** *Let S be a set of nodes that is connected to $C_i \setminus S$ for some cluster C_i by volume of at least $6|S| \cdot M$. Then at least $|S|M$ volume can be moved from S to $C_i \setminus S$.*

▶ **Lemma 9.** *Let X_t' be a subsequence of transactions where transactions incident to C_i have volume at least $tr \geq M$, then Algorithm 3 can transport at least tr from any set of nodes $S \subset C_i$ to any other set $T \subset C_i$. The LP solution is modified by volume at most $\mathcal{O}(\sqrt{p} \cdot tr)$.*

We now have the requisite ingredients to state and prove (in Section A.7) the main theorem of this section.

▶ **Theorem 10.** *Algorithm 2 is $\mathcal{O}(\sqrt{p})$ approximation of the problem where given a transaction sequence, the nodes are $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered.*

We also get the following corollary as a direct consequence of the proof of Theorem 10.

▶ **Corollary 11.** *Given (m, k, t)-clustered transaction sequence where $t \geq 24\sqrt{p}$, Algorithm 2 is $\mathcal{O}(\max(m, k))$-approximation of the problem.*

Finally, we end this section with two observations. The first observation effectively states that our algorithm just needs to focus on the total volume of partially accepted transactions and can ignore the volume of fully accepted and rejected transactions. The second observation allows us to further reduce the approximation ratio if we assume clustering of a larger depth.

Practical relaxations. We observe that the proof of Theorem 10 requires a condition on the partially accepted transactions (these are transactions which satisfy $0 < y_i < x_i$). However, we stress that the volume of fully accepted or rejected transactions can be arbitrary. Moreover, in every cluster, the middle node can interact with other clusters arbitrarily and this also does not affect the proof of Theorem 10.

Hierarchical clustering. We can use a similar idea to further reduce the approximation ratio if we assume that within each cluster the clusters themselves are also clustered. Namely, we first treat the top-level clusters as nodes and solve the problem as outlined by Algorithm 1. Then, we have some transactions pre-accepted or pre-rejected and we now shift the focus to the next level clusters. If these next-level clusters consist of other clusters, we treat them

again as nodes. We do this recursively until we reach the lowest-level clusters and then use Algorithm 2 to solve the problem within the lowest-level clusters. Note that we require the strength of the clustering to be bigger in order to achieve a similar approximation ratio in this setting. Nevertheless, assuming the nodes satisfy sufficiently strong clustering conditions, we can achieve an approximation ratio of $\mathcal{O}(p^{\frac{1}{3}})$ for two nested clusterings, and, in general, $\mathcal{O}(p^{\frac{1}{k+1}})$ for k nested clusterings (if the nodes in clusters are well distributed). We stress that although hierarchical clustering allows us to achieve a stronger approximation factor, it also comes with additional structural assumptions. However, we believe there are strong use cases of hierarchical clustering, for instance using a payment channel network to handle payments among users in a large company. The clusters would naturally follow the clustering of users into divisions and sub-divisions, with the head of each division being the centre of the star in their corresponding cluster.

6 Case study: the Lightning Network

To justify our transactions distribution assumption in Section 5, we perform an empirical case study of the largest and most commonly used PCN: Bitcoin's Lightning Network. Specifically, the research question we want to investigate is the following:

Are transactions in the Lightning Network clustered according to the clustering conditions stated in Section 5?

To address this question, we obtained the latest snapshot (dated September 2023) of the Lightning Network from the Lightning Network gossip repository [1]. We further restrict our examination to the largest connected component which contains $20,348$ users and $310,657$ channels. Figure 5 shows the distribution of degrees in the largest connected component, where a large number of nodes ($\sim 10\%$) are of degree ≤ 2.

Estimating the source and target of transactions. The main challenge in extracting transactions data from the Lightning Network is that the Lightning Network, by design, only provides information about the costs of transactions across channels, but does not provide information about transactions between users. Nevertheless, it is reasonable to assume that transactions between users are related to the channel capacities that connect them. Furthermore, we are mainly concerned with finding out if the clustering conditions hold for transactions between ordinary users with low degrees rather than between, e.g., large payment hubs with high degrees. We hence propose the following methodology: we first remove high degree nodes (i.e., nodes with degree ≥ 3) in the network and connect their neighbours to each other. Let us denote this resulting reduced network by G. In G, we assume nodes that are directly connected to each other transact with each other. Our methodology of generating the reduced network G is detailed in Algorithm 4 in our extended technical report [8].

Estimating the volume of transactions. Recall that a crucial clustering condition is that the volume of transactions that stay within a cluster is larger than the volume of the transactions that go between clusters. However, the privacy principles behind the design of the Lightning Network renders such transaction volume information hidden. Thus, a second challenge in our setting is extracting estimates of the size of transactions between users. In order to estimate transaction volume, we assume that the volume of transactions between two connected users in G is inversely proportional to the cost of a transaction. Namely, that the volume of transactions between users i and j is $1/(c_{i,j}+1)$ where $c_{i,j}$ is the minimum cost of sending a transactions of 100 milli-satoshi from i to j.

Note that the volume of transactions is estimated under the simplifying assumptions that all transactions have the same fixed volume, and therefore the cost for a transaction between a pair of users is fixed and specified using $cost_H$ on input.

We start with the largest strongly connected component of H as it is the largest set of users such that any user can send a transaction to any other. Also, to simplify the experiment, we assume that the volume of transactions from a user u to another user v is the same as the volume of transactions in the reverse order (from v to u).

This approach of computing costs for each pair of users runs in time linear with the number of edges of G. Note that the number of updates of a cost is at most the number of edges adjacent to the set $users$, which is bounded by $O(|users|)$ as the degree of each vertex is bounded.

Results. Figure 6 and Table 1 in our extended technical report [8] show that we obtain 18 clusters with average within-cluster transaction volume of 826.1, and between-cluster transaction volume of 4.27. This gives us a ratio of 193.2 of within-cluster and between-cluster transactions, which confirms our experimental hypothesis.

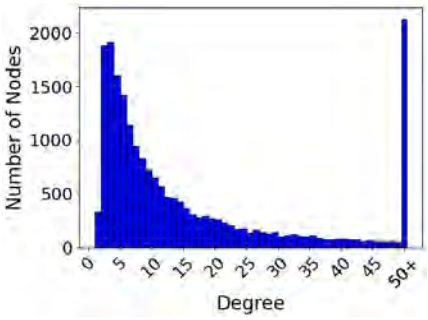


Figure 5 Degree distribution of the nodes in the largest connected component of the Lightning Network.

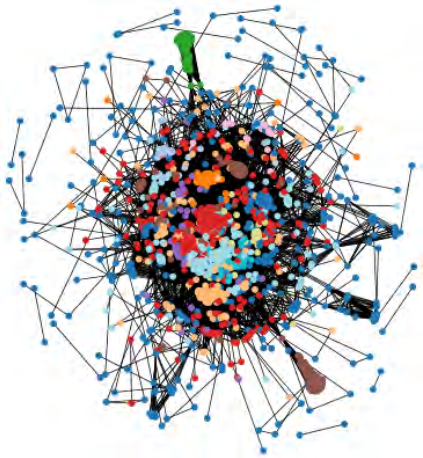


Figure 6 Clustering of the nodes in the reduced network G. Each color represents a different cluster.

7 Conclusion

In this work, we generalise a model from processing transactions over a single payment channel to an entire PCN. To this end, we first define a network creation stage of the problem which examines how to create a network with minimal capacity cost, and we show that the star is a 2-approximation of the optimal capacity cost. We then propose two algorithms to address weighted transaction processing over a network. The first algorithm assumes a star topology over the p parties, but apart from that works without any further assumptions on the transaction sequence, and gives an approximation ratio of $\mathcal{O}(p)$. The second algorithm assumes a double star topology and a clustering of the transaction sequence, and gives an approximation ratio of $\mathcal{O}(\sqrt{p})$. Finally, we also perform an empirical case study estimating transaction information from the topology of the Lightning Network, and our estimates suggest that our clustering assumptions regarding the input transaction sequence are close to reality.

We believe our work is an important first step in the direction of designing optimal networks and algorithms for transaction processing in PCN. We highlight two interesting directions for future work. First, it would be interesting to extend our model and algorithms to the online setting, which will also complement the analysis of [4, 7]. Secondly, developing better estimates regarding transaction information in real world PCNs would also be paramount to developing better optimisation algorithms and heuristics.

References

1 Lightning network topology datasets. URL: https://github.com/lnresearch/topology.

2 Raiden network. https://raiden.network/, 2017.

3 Hagit Attiya, Leah Epstein, Hadas Shachnai, and Tami Tamir. Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica*, 57(1):44–61, 2010. doi:10.1007/S00453-008-9195-X.

4 Georgia Avarikioti, Kenan Besic, Yuyi Wang, and Roger Wattenhofer. Online payment network design. In *DPM/CBT@ESORICS*, volume 11737 of *Lecture Notes in Computer Science*, pages 307–320. Springer, 2019. doi:10.1007/978-3-030-31500-9_20.

5 Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the lightning: The game theory of payment channels. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 264–283. Springer, 2020. doi:10.1007/978-3-030-51280-4_15.

6 Zeta Avarikioti, Tomasz Lizurej, Tomasz Michalak, and Michelle Yeo. Lightning creation games. *CoRR*, abs/2306.16006, 2023. doi:10.48550/arXiv.2306.16006.

7 Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali Maddah-Ali, Stefan Schmid, Jakub Svoboda, and Michelle Yeo. R2: boosting liquidity in payment channel networks with online admission control. In *FC (1)*, volume 13950 of *Lecture Notes in Computer Science*, pages 309–325. Springer, 2023. doi:10.1007/978-3-031-47754-6_18.

8 Krishnendu Chatterjee, Jan Matyáš Křišťan, Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Boosting payment channel network liquidity with topology optimization and transaction selection. Cryptology ePrint Archive, Paper 2025/1484, 2025. URL: https://eprint.iacr.org/2025/1484.

9 Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and scalability. In *QRS Companion*, pages 122–128. IEEE, 2018. doi:10.1109/QRS-C.2018.00034.

10 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 156–165, 2004. doi:10.1145/1007352.1007383.

11 Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Struct. Algorithms*, 18(2):116–140, 2001. doi:10.1002/1098-2418(200103)18:2\%3C116::AID-RSA1001\%3E3.0.CO;2-2.

12 Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (A position paper). In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2016. doi:10.1007/978-3-662-53357-4_8.

13 Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. https://blockstream.com/eltoo.pdf, 2018.

14 Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems (SSS)*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-21741-3_1.

15 Oguzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to profit from payments channels. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 284–303. Springer, 2020. doi:10.1007/978-3-030-51280-4_16.

16 Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Topology. *SSRN Electronic Journal*, 2023. doi:10.2139/ssrn.4439190.

17 Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Channels. *Manag. Sci.*, 70(6):3827–3840, 2024. doi:10.1287/MNSC.2022.01664.

18 Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Polymorphic contention management. In Pierre Fraigniaud, editor, *Distributed Computing*, pages 303–323, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11561927_23.

19 Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Towards a theory of transactional contention managers. In *PODC*, pages 316–317. ACM, 2006. doi:10.1145/1146381.1146429.

20 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *PODC*, pages 307–316. ACM, 2019. doi:10.1145/3293611.3331589.

21 Piyush Kumar Gupta and Panganamala Ramana Kumar. The capacity of wireless networks. *IEEE Trans. Inf. Theory*, 46:388–404, 2000. doi:10.1109/18.825799.

22 Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991. doi:10.1145/114005.102808.

23 Paul Holland, Kathryn B. Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5:109–137, 1983. URL: https://api.semanticscholar.org/CorpusID:34098453.

24 Anurag Jain, Shoeb Siddiqui, and Sujit Gujar. We might walk together, but I run faster: Network fairness and scalability in blockchains. In *AAMAS*, pages 1539–1541. ACM, 2021. doi:10.5555/3463952.3464152.

25 Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration outsourcing for state channels. In *AFT*, 2019. URL: 10.1145/3318041.3355461.

26 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2015.

27 Prabhakar Raghavan and Clark D Thompson. Provably good routing in graphs: regular arrays. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 79–87, 1985. doi:10.1145/22145.22154.

28 Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Weighted packet selection for rechargeable links in cryptocurrency networks: Complexity and approximation. In *SIROCCO*, volume 13892 of *Lecture Notes in Computer Science*, pages 576–594. Springer, 2023. doi:10.1007/978-3-031-32733-9_26.

A Omitted proofs

A.1 Proof of Lemma 1

▶ **Lemma 1.** *Given the sequence of transactions $\mathcal{S}$, we have $LP(\mathcal{S}, K_p) \leq AC(\mathcal{S}, G)$.*

Proof. We first observe that adding channels to a graph can only decrease the capacity cost (adding channels can only make the shortest paths shorter). This means that the capacity cost is the lowest for the complete graph, no matter what is the optimal graph topology (as computed by minimising the *total* cost of both capacity and channel creation). The LP computes the optimal capacity for the complete graph K_p, which is a superset of every graph, including the optimal graph. ◀

A.2 Proof of Theorem 2

▶ **Theorem 2.** *For every graph G over p nodes, and any graph G' with diameter d, the following inequality holds*

$$AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)\,.$$

Proof. Since for the complete graph K_n, we have $AC(\mathcal{S}, K_n) \leq AC(\mathcal{S}, G)$, we simply need to show that $AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)$. We modify the solution of the LP for the complete graph by adding at most d-times the capacity and prove that this graph processes all transactions.

We start with graph G' with zero capacities. For all pairs of nodes v and u, let the capacity between them be $w_{v,u}$. We find the shortest path between them in G', since the diameter is at most d, the length of the path is at most d. Then, we increase the capacity of every channel on the path by $w_{v,u}$. We denote this capacity to a special bucket that will be used to process transactions that the LP sends along u, v. At the start at edge k, l, where k is closer to v and l is to u, we increase the balance at the k's side by $C^0_{v,u}$ (and l's balance is $C^0_{u,v}$), the balance at the edge v, u at the start as given by the LP.

After we increase the capacities of G' for every pair of nodes, we process the transactions. Any time the transaction of size s was forwarded along some edge u, v, we can transfer s on the shortest path between u and v from the bucket on the edge denoted u, v. If the bucket for some edge becomes negative, the solution of LP is infeasible.

By this, we can process all transactions. Since the diameter is at most d, we know that the increase in capacity is at most d-times. ◀

A.3 Proof of Theorem 6

▶ **Theorem 6.** *Given the transaction sequence X_t, let $\mathcal{C}$ be the optimal capacity cost and $\mathcal{R}$ the optimal rejection cost on a star. Then Algorithm 1 run for every pair of nodes incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$.*

Proof. Every node has $p-1$ connections while the size of the channel is viewed as the minimum of the two sizes as denoted by the LP. The reserves are $\sqrt{3}$ times that value. Moreover, every channel in a star has funds amount as dictated by the LP. Altogether it gives $1+(p-1)\cdot\sqrt{3}$ times of $\mathcal{C}$.

The algorithm ensures that the in every channel on the right side, there is at least the amount prescribed by the LP. This means that all fully accepted ($x_i = y_i$) transactions are accepted. Moreover, from [28], we know that the rejection cost between two nodes is at most $\sqrt{3}+1$ times the rejection cost of the optimal rejection cost $\mathcal{R}$. Thus, Algorithm 1 incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$. ◀

A.4 Proof of Lemma 7

▶ **Lemma 7.** *If there is a transaction (s, t, x_i, y_i), we can change it to*

- *$(s, t, x_i, y_i + \alpha)$ to simulate moving α reserves from s to t, and*
- *$(s, t, x_i, y_i - \alpha)$ to simulate moving α reserves from t to s.*

Proof. Every node has $p-1$ connections while the size of the channel is viewed as the minimum of the two sizes as denoted by the LP. The reserves are $\sqrt{3}$ times that value. Moreover, every channel in a star has funds amount as dictated by the LP. Altogether it gives $1+(p-1)\cdot\sqrt{3}$ times of $\mathcal{C}$.

The algorithm ensures that the in every channel on the right side, there is at least the amount prescribed by the LP. This means that all fully accepted ($x_i = y_i$) transactions are accepted. Moreover, from [28], we know that the rejection cost between two nodes is at most $\sqrt{3}+1$ times the rejection cost of the optimal rejection cost $\mathcal{R}$. Thus, Algorithm 1 incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$. ◀

A.5 Proof of Lemma 8

▶ **Lemma 8.** *Let S be a set of nodes that is connected to $C_i \setminus S$ for some cluster C_i by volume of at least $6|S| \cdot M$. Then at least $|S|M$ volume can be moved from S to $C_i \setminus S$.*

Proof. From the condition 3 of the clustering conditions, we know that at least $2|S|M$ volume moves from the cluster and $2|S|M$ volume to the cluster. For the sake of contradiction, suppose that we can move at most $m < |S|M$ from S to $C_i \setminus S$.

From Lemma 7, we know that we cannot move only on fully accepted transactions from S and rejected transactions from $C_i \setminus S$. Let m' be the incoming volume to S that can be moved. Then in the subsequence in the solution of LP, S receives m' volume and gives up $2|S|M - (m - m')$ volume. That means on average it lost at least $2|S|M - (m - m') - m' = 2|S|M - m > |S|M$. From the Dirichlet principle, at least one node lost more than M. This is impossible since M is the highest capacity in C_i and the solution of the LP ensures that for any subsequence the difference is at most M. ◀

A.6 Proof of Lemma 9

▶ **Lemma 9.** *Let X'_t be a subsequence of transactions where transactions incident to C_i have volume at least $tr \geq M$, then Algorithm 3 can transport at least tr from any set of nodes $S \subset C_i$ to any other set $T \subset C_i$. The LP solution is modified by volume at most $\mathcal{O}(\sqrt{p} \cdot tr)$.*

Proof. Since there was tr transported, inside the cluster there was at least $24 \cdot tr \cdot |C_i|$ transported, and next to S, there was at least $6tr|S|$ transported. From Lemma 8, that means that at least $tr|S|$ goes away from S to other nodes.

Let $S_0 = S$ and S_i be the nodes that we can transport nonzero amount from S_{i-1}. Similarly, let $T_0 = T$ and T_i be the nodes that we can transport nonzero amount to T_{i-1}. Since the clustering conditions, we have that the size of S_i and T_i is increasing if their size is at most half of the cluster. That means they intersect at some point.

We look at the minimum mn that can be transported and we change the transactions such that in v the reserves are decreased by mn and in u they are increased by mn. After the change, we know that the clustering conditions still hold, now we just need to decrease the inequalities by mn. Therefore, we can find the path again. ◀

A.7 Proof of Theorem 10

▶ **Theorem 10.** *Algorithm 2 is $\mathcal{O}(\sqrt{p})$ approximation of the problem where given a transaction sequence, the nodes are $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered.*

Proof. We are using a double star, which means it is at least a 4 approximation of the optimal topology from Theorem 2.

The algorithm uses Algorithm 1 for every pair of clusters (between middle nodes) to accept and reject transactions between them. This incurs cost for additional capacity equal $\mathcal{O}(\sqrt{p})$ times the cost of the capacity cost of the LP for the capacity between clusters. The rejection cost is at most $(\sqrt{3}+1)$ times higher than the rejection cost between clusters. All of this follows from Theorem 6.

Next, from Lemma 9, we know that we can transport reserves such that everything between clusters is accepted or rejected from the previous algorithm. It changes the values of the LP by at most $\mathcal{O}(\sqrt{p})$ times the volume of rejected transactions between clusters.

Finally, again from Theorem 6, we can solve the problem inside the clusters. This incurs capacity cost $\mathcal{O}(\sqrt{p})$ times the capacity cost of the solution inside the clusters. Moreover, the cost for rejection is $(\sqrt{3}+1)$ times the rejection cost in the modified solution of LP (X'_t).

The modified rejection cost of solution of LP is the rejection cost of the original solution plus at most $\mathcal{O}(\sqrt{p})$ times the rejection cost of transactions between clusters (from Lemma 9). Altogether, we get that the algorithm is $\mathcal{O}(\sqrt{p})$ approximation algorithm. ◀

B Omitted details of star topology solution

B.1 Modified channel algorithm details

We now describe the decision making process in Algorithm 1 for transactions going from u to v (the other direction is symmetric). Algorithm 1 accepts a transaction as long as the reserve R_u is above a threshold (i.e., $R_u \geq \frac{\sqrt{3}-1}{2}M$, line 6). Accepting a transaction decreases the capacities C_u and R_u^v of u by y_i and $x_i - y_i$ respectively (refer to the "accept" case in Figure 2). If accepting the transaction would make R_u decrease by too much and the LP demands to accept the transaction *weakly* (i.e., $\frac{y_i}{x_i} < \frac{\sqrt{3}}{\sqrt{3}+1}$, line 9), the algorithm rejects the transaction. Rejecting a transaction from say u to v decreases both C_u and R_u^v of by y_i (refer to the "reject" case in Figure 2). If the reserve on u's side after accepting the transaction would be too small (i.e., $R_u < \frac{\sqrt{3}-1}{2}M$, line 17), the algorithm looks at all subsequent transactions *between* u and v in the transaction sequence. It accepts all transactions that increases R_u while rejecting weakly accepted transactions from u to v (rejecting them also increases R_u). From *strongly* accepted transactions (i.e., $\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{\sqrt{3}+1}$) from u to v, the algorithm tries to accept all of them while simulating the movement of capacities between the reserves until either $R_u > \frac{\sqrt{3}-1}{2}M$ or $R_u < 0$. If $R_u > \frac{\sqrt{3}-1}{2}M$, then all transactions can be accepted. If $R_u < 0$, the algorithm rejects strongly accepted transactions until $R_u > \frac{\sqrt{3}-1}{2}M$. The proof in [28] shows that doing so maintains the approximation ratio at $\sqrt{3}+1$ for the rejection cost.

Algorithm details. The simulating of the reserves is implemented by a heap. The heap is empty while $R_u \geq \frac{\sqrt{3}-1}{2}M$, and when $R_u < \frac{\sqrt{3}-1}{2}M$ it starts to fill up. When $R_u < 0$, then the decision is changed for transactions from the top of the heap until $R_u \geq \frac{\sqrt{3}-1}{2}M$ again.

Observe that for Algorithm 1 the funds on both sides can shift and the reserves are not affected. This allows us to compose the algorithm from pairwise reserves with shared funds denoted by the linear program.

The algorithm describes decisions for the star, so a transaction goes from u to c (star center) and then from c to v. That means it needs to traverse two channels. The algorithm sees it as only pairwise interaction. Since the decisions for the two channels are always the same for both channels (a transaction is either accepted or rejected for both), we can have the reserves stored twice, once between u and c, the second time between v and c. The value R_u is then the reserves at u in channel u, c and at c in the channel c, v. Both of these reserves move in sync.

C Omitted details of clustering algorithm

C.1 Stochastic Block Model details

The planted partition model creates a random graph where nodes are partitioned into clusters. Inside a cluster, two nodes are connected with probability q_1, otherwise, they are connected with probability q_2. The planted partition model creates an unweighted and undirected graph. Our graph created by the packets is directed, weighted, and temporal. These changes call for modification of the planted partition model.

The biggest change to the model is brought by the temporality. Some edges follow after others, we require that all of the conditions hold for some continuous subsequence of edges incident to some cluster of bigger size.

To make the graph weighted, we view q_1 and q_2 as ratios between volumes of the packets between the clusters and inside the cluster. This view is expressed by the condition 1 of the clustering conditions.

To make the graph directed, we allow for arbitrary orientation of the edges with the condition that at least $\frac{1}{3}$ of the volume needs to be directed in both directions. It is expressed by condition 2 of the clustering conditions. Note that $\frac{1}{3}$ of the volume can be oriented in any direction, no matter the other packets.

Finally, the condition 2 of the clustering conditions ensures that some volume is incident to every node.

Note that all of the conditions would be satisfied with high probability in the stochastic block model. Moreover, there are other conditions that would be satisfied (for instance any two clusters would send packets between each other at the same rate) that we do not require. From the above the graph generated by the planted partition model with $\sqrt{p}$ partitions of size $\sqrt{p}$ and $q_2 = \frac{1}{p}$ and $q_1 = 1 - q_2$ satisfies all clustering conditions with high probability.

C.2 Linear program for double star

Let the set of leaf nodes be L and the set of middle nodes be M, and c denote the central node of the star. For a leaf node u, let $M(u)$ be the middle node corresponding to u's cluster.

For a leaf node v and natural number i, we use $C_{v,i}$ and $C'_{v,i}$ to denote the capacity at the channel $(v, M(v))$ after processing the i-th transaction on the side of v and $M(v)$ respectively. Moreover let C_v denote the total capacity of the channel $(v, M(v))$, i.e., $C_v = C_{v,i} + C'_{v,i}$. For a middle node v and i, we use $M_{v,i}$ and $M'_{v,i}$ to denote the capacity at the channel (v, c) after processing the i-th transaction on the side of v and c respectively. Moreover, let M_v denote the total capacity of the channel (v, c).

$$\text{minimise} \sum_{u \in L} C_u + \sum_{u \in M} M_u + \tag{2}$$

$$\sum_i f(x_i - y_i) + \frac{m(x_i - y_i)}{x_i} \tag{3}$$

$$\begin{aligned}
\forall i &: 0 \le y_i \le x_i \\
\forall (v, u, x_i) &: C_v, C_u \ge x_i \\
&\quad C_{v,i} = C_{v,i-1} - y_i \\
&\quad C'_{v,i} = C'_{v,i-1} + y_i \\
&\quad C'_{u,i} = C'_{u,i-1} - y_i \\
&\quad C_{u,i} = C_{u,i-1} + y_i \\
\forall (v, u, x_i) \text{ between clusters:} &\ M_{M(u)}, M_{M(v)} \ge x_i \\
&\quad M_{M(v),i} = M_{M(v),i-1} - y_i \\
&\quad M'_{M(v),i} = M'_{M(v),i-1} + y_i \\
&\quad M'_{M(u),i} = M'_{M(u),i-1} - y_i \\
&\quad M_{M(u),i} = M_{M(u),i-1} + y_i
\end{aligned}$$

Algorithm 3 Transporting reserves between nodes.

Input: Subsequence of transactions X_t inside one cluster, two nodes v, u, amount A.
Output: Modified X_t such that A in reserves is transported from v to u.

```
1: procedure TRANSPORT(X_t, v, u, A)
2:     M ← PATHBETWEEN(v, u)
3:     mn ← MINIMUMONPATH(M)
4:     mn ← min(mn, A)
5:     for (dir, x, y, x_i, y_i) ∈ M do
6:         if dir from v to u then
7:             y_i ← y_i − mn
8:         else
9:             y_i ← y_i + mn
10:    if A − mn > 0 then
11:        TRANSPORT(X_t, v, u, A − mn)
```

C.3 Algorithm to transport cluster reserves

Here we detail Algorithm 3, which transfers cluster reserves from well-funded to depleted nodes *within* a cluster, so that the cluster nodes have sufficient reserves to handle transactions that go *between* the clusters.

Validity in Network-Agnostic Byzantine Agreement

Andrei Constantinescu
ETH Zürich, Switzerland
DFINITY Foundation, Zürich, Switzerland

Marc Dufay
ETH Zürich, Switzerland

Diana Ghinea[1]
Lucerne University of Applied Sciences and Arts, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

Byzantine Agreement (BA) considers a setting of n parties, out of which up to t can exhibit byzantine (malicious) behavior. Honest parties must decide on a common value (agreement), which must belong to a set determined by the honest inputs (validity). Depending on the use case, this set can grow or shrink, leading to various possible desiderata collectively known as validity conditions. Varying the validity property requirement can affect the regime under which BA is solvable.

Our work investigates how the selected validity property impacts BA solvability in the network-agnostic model, where the network can either be synchronous with up to t_s byzantine parties or asynchronous with up to $t_a \leq t_s$ byzantine parties. We give necessary and sufficient conditions for a validity property to render BA solvable, both for the case with cryptographic setup and for the one without. This traces the precise boundary of solvability in the network-agnostic model for every validity property. Our proof of sufficiency provides a universal protocol, that achieves BA for a given validity property whenever the provided conditions are satisfied.

We note that, for any non-trivial validity property, the condition $2 \cdot t_s + t_a < n$ is necessary for BA to be solvable, even with cryptographic setup. Specializing this claim to $t_a = 0$ gives that $t < n/2$ is required whenever one expects a purely synchronous protocol to also work in an asynchronous network when there are no corruptions. This is especially surprising given that, for some validity properties, $t < n$ is a sufficient condition without the last stipulation.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases byzantine agreement, validity, network-agnostic protocols

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.24

Related Version *Full Version*: `https://arxiv.org/abs/2410.19721` [9]

1 Introduction

Achieving agreement among the parties involved in a distributed system is crucial for maintaining consistent views. This becomes particularly challenging due to the potential for parties' failures, which can range from benign crashes to malicious (byzantine) behavior. Byzantine Agreement (BA) is an extensively studied problem in distributed computing that tackles this challenge. It seeks to establish a common value amongst a set of n parties even when up to t parties exhibit byzantine behavior. A crucial aspect of BA lies in its validity condition, which requires that the value agreed upon reflects the honest parties' proposals rather than being a default or arbitrary value. Standard BA definitions consider the so-called *strong unanimity* (also known as *strong validity*): if all honest parties propose the same value v,

[1] This work was partially carried out while the author was at ETH Zürich.


© Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0



the agreed-upon output must be v. This is a powerful guarantee in applications concerning binary decisions, as it coincides with *honest-input validity*, which requires agreement on an honest input. However, strong unanimity fails to provide meaningful outputs for larger input spaces. For instance, consider a set of parties running a BA protocol to agree on a room's temperature: minor measurement errors are inherent, and hence strong unanimity allows agreement on a corrupted party's proposal. Similar issues arise in scenarios such as deciding on a location using GPS coordinates [5] or when the map is modeled as a graph [11, 30].

While achieving honest-input validity in scenarios where the input space is large (size $\omega(n)$) is impossible [29], the literature offers a plethora of weaker alternatives that are stronger and more meaningful than strong unanimity. For instance, one may avoid corrupted outputs by enhancing strong unanimity with an additional condition called *intrusion tolerance* [13, 22, 23]: the honest parties either agree on an honest input or on a special symbol $\perp$. In the aforementioned scenarios of deciding on a measurement or a meeting point, a highly suitable alternative is the so-called *convex validity*: the output agreed upon must be in the honest parties' inputs convex hull, i.e., within the range of honest inputs if the input space is a subset of $\mathbb{R}$ [11, 22, 23, 28, 30, 32]. Stronger variants for real values require the outputs to be close to the honest inputs' median [10, 31], or to the k-th lowest honest input [27]. However, the honest-range approach is not a universal solution: this would carry no meaning in a voting problem if we represented candidates with integers. Instead, approaches based on social choice theory (*Pareto validity*) lead to more appropriate validity definitions [26].

The multitude of validity definitions leads to a natural question: what are the necessary and sufficient conditions for achieving BA with a given validity property, i.e., *to solve a given validity property*? This question can be concerned with multiple aspects, such as resilience thresholds t, round complexity, or message complexity. In addition, the conditions may depend on whether a cryptographic setup and randomization are available. The communication model assumed also plays an important role: one extreme is the synchronous model, where all parties have synchronized clocks, and all messages get delivered within a predefined amount of time Δ. This enables elegant protocols that operate in rounds, but that may fail in a real-life network where sporadic issues are possible. The other extreme is the asynchronous model, which only assumes that messages get delivered eventually. The asynchronous model comes with highly robust protocols, but also with important drawbacks: **(a)** lower resilience threshold: e.g., $t < n/3$ as opposed to $t < n/2$ (assuming digital signatures) for strong unanimity, or $t < n/4$ as opposed to $t < n/3$ for convex validity in $\mathbb{R}^2$; **(b)** randomization is a requirement [19]. The middle ground between the two models has also been considered. The partially synchronous model [17] bridges the gap between the two extremes by assuming that the network is initially asynchronous and eventually becomes synchronous. In the synchronous and partially synchronous models, the solvability of validity conditions using deterministic protocols is completely understood [7, 8].

In this work, we are concerned with a different paradigm for bridging the gap between synchrony and asynchrony, namely the *network-agnostic* model [4]: parties are initially unaware of whether the network is synchronous or not: if it is synchronous, up to t_s of the parties involved may be corrupted, and otherwise only up to $t_a \leq t_s$. Network-agnostic protocols are designed to provide guarantees in both cases. We ask the following question:

Under what conditions can a validity property be solved in the network-agnostic model?

1.1 Our Contribution

We provide a complete characterization for achieving network-agnostic BA with a given validity property, establishing tight necessary and sufficient conditions.

We first show that, even if cryptographic setup is available, the condition $n > 2 \cdot t_s + t_a$ is a requirement for any non-trivial validity condition to be solvable (i.e., a condition for which simply outputting a default value does not suffice). When no cryptographic setup is available, we show the stronger requirement of $n > 3 \cdot t_s$. Our proof for the latter, in fact, works in the synchronous model and, therefore, strengthens the characterization provided by [7] for the synchronous model. In particular, [7] only focuses on deterministic protocols, while our proofs rely on different techniques that also apply to randomized protocols.

Afterwards, regardless of whether cryptographic setup is available or not, we add one more necessary condition, which is an adaptation of the *similarity condition* of [8] and the *containment condition* of [7] to the network-agnostic model. Roughly, this requires that *similar* honest inputs' configurations have a valid output in common. The term *similar* captures that some of the proposed inputs may come from corrupted parties, and that, in asynchronous networks, some honest inputs may be missing due to high network delays.

Finally, we show that, together, the aforementioned conditions are also sufficient by providing a universal protocol, that achieves network-agnostic BA for a given validity property whenever these conditions are satisfied. This is a more general variant of the protocol of [11]. In particular, the requirement for solvability is precisely $n > 2 \cdot t_s + t_a$ together with the similarity condition assuming cryptographic setup, and $n > 3 \cdot t_s$ together with the similarity condition assuming no cryptographic setup. Our protocol is randomized, which is a requirement when the network may be asynchronous and $t_a > 0$ [19].

1.2 Related Work

General validity conditions. The foundational investigation into general validity properties was initiated by Civit et al. [8] for the partially synchronous model. Subsequently, Civit et al. [7] embarked on a follow-up study, extending their analysis to the synchronous model. Both works provide a complete characterization, identifying the necessary and sufficient conditions for solving a validity property deterministically. We also note that the contributions of [7, 8] extend beyond this characterization, an important side of these works lying in the exploration of lower bounds on message complexity. This generalizes the well-established Dolev-Reischuk bound on message complexity for BA with strong unanimity [15] to encompass the broader landscape of non-trivial validity properties. Considerations of message complexity are outside our scope. We also note that *probabilistic* validity properties are outside the scope of the analysis of Civit et al, and also outside our scope. A notable example is *qualitative validity*, introduced by Goren et al [25].

Network-Agnostic BA and particular validity conditions. Designing protocols that achieve security guarantees in both synchronous and asynchronous networks has been the subject of an extensive line of work. The network-agnostic paradigm was introduced by Blum, Katz and Loss [4]. The work of [4] shows that, if a public key infrastructure is provided, BA with strong unanimity can be achieved if and only if $n > 2 \cdot t_s + t_a$. Further works on network-agnostic BA with strong unanimity have focused on improving the efficiency guarantees [12, 13].

Due to its broad applicability, convex validity within the network-agnostic communication paradigm has attracted increased attention. Ghinea, Liu-Zhang and Wattenhofer [20, 21] have investigated the feasibility of achieving convex validity for a weaker variant of BA, known as Approximate Agreement [1, 14]. In particular, [20] shows that Approximate Agreement on real numbers is solvable under the same necessary and sufficient condition $n > 2 \cdot t_s + t_a$ assuming a public key infrastructure. Building on the previous, [21] gives sufficient conditions for the multidimensional variant of the problem that match the known requirements in the

pure synchronous and asynchronous models [28, 32]. Returning to the non-approximate version, Constantinescu et al. [11] have provided the tight conditions for network-agnostic BA with convex validity for abstract convex spaces. In this case, the conditions include $n > 2 \cdot t_s + t_a$ or, if no cryptographic setup is available, $n > 3 \cdot t_s$, along with a few additional conditions that depend on the Helly number of the convexity space.

Other takes on network-agnostic BA have been considered, such as *scaling* the validity guarantees with the network conditions: for real-valued inputs, [10] proposes a protocol for BA with median validity guaranteeing that the output is closer to the honest inputs' median when the network is synchronous than when it is asynchronous. This protocol simultaneously matches the optimal closeness guarantees for purely synchronous [27, 31] and purely asynchronous [10] networks. Such validity properties that scale with the network conditions are outside our scope.

Comparison to previous works. As outlined above, the conditions $n > 2 \cdot t_s + t_a$ and, if no cryptographic setup is available, $n > 3 \cdot t_s$, have been proven to be necessary for strong unanimity properties, i.e., stronger than strong unanimity [4]. Our work shows that this is a requirement for weaker validity properties as well, i.e., for any non-trivial property. We find this result surprising especially for *weak validity*: if *all parties are honest* and hold input v, then the output agreed upon must be v. Assuming a public key infrastructure, this property is solvable in the synchronous model for $t_s < n$, as a straightforward application of the Dolev-Strong broadcast protocol [16]. On the other hand, our result implies that if we expect a BA protocol with weak validity to remain secure in the asynchronous model even for no corruptions (i.e., $t_a = 0$), then the synchronous resilience threshold steps down from $t_s < n$ to $t_s < n/2$.

In contrast to the work of [11] regarding network-agnostic BA with convex validity, our results move the difficulty of proving such feasibility results as a whole to only verifying whether a validity condition satisfies our similarity condition. That is, one can show that convex validity satisfies this similarity condition if and only if the necessary Helly number-based conditions of [11] hold. Our impossibility arguments diverge: we investigate these under *any* validity property, while the work of [11] considers a fixed validity property but also shows impossibility under weaker agreement requirements. On the other hand, our protocol matching our lower bounds is a more general variant of the protocol of [11].

Our paper provides a characterization similar to those of [7, 8] for the synchronous and partially synchronous settings, respectively. The key difference is that these two models allow for deterministic protocols, while the network-agnostic model inherently requires randomization for achieving BA when $t_a > 0$ [19]. Consequently, the focus shifts towards randomized protocols, requiring our proofs to employ different techniques. While the arguments behind the *containment* condition of [7] can be easily adapted for randomized protocols, this is not immediate for their proof that $n > 3 \cdot t_s$ is necessary when no cryptographic setup is available. Our proof for this lower bound, in fact, assumes the synchronous setting and, therefore, strengthens the characterization of [7]. Summing up, their necessary conditions now hold even for randomized protocols, and, as shown in their paper, they can be matched by deterministic protocols.

2 Preliminaries

We consider a setting with n parties $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ running a protocol in a fully-connected network, where links model authenticated channels. We will be working in the *network-agnostic* model: the network may be synchronous, or asynchronous, and the parties

are not aware a priori of the type of network. If the network is synchronous, the parties hold perfectly synchronized clocks and each message is delivered within a publicly known amount of time Δ. Otherwise, if the network is asynchronous, messages can get delayed for an arbitrary amount of time, and clocks may not be synchronized.

Adversary. We assume a central adversary that may corrupt up to t_s of the n parties if the network is synchronous, and up to $t_a \leq t_s$ parties if the network is asynchronous. Corrupted parties permanently become byzantine, and hence may deviate arbitrarily (maliciously) from the protocol. The adversary additionally controls the message delivery schedule, subject to the conditions of the network type. Our impossibility results assume a static adversary (i.e., chooses which parties to corrupt at the beginning of the protocol's execution). Our protocol, on the other hand, provides security even against an adaptive adversary (i.e., chooses which parties to corrupt at any point in the protocol's execution).

Cryptographic setup. We will consider both settings with and without cryptographic setup. Protocols assuming a cryptographic setup will make use of a public key infrastructure (PKI) and a secure signature scheme, and only hold against a computationally bounded adversary. For simplicity of presentation, we assume that the signatures are perfectly unforgeable.

Byzantine Agreement and Validity. A Byzantine Agreement (BA) protocol assumes that each (honest) party holds a value $v_{\text{IN}} \in V_{\text{IN}}$ as input, and enables the parties to agree on a common output $v_{\text{OUT}} \in V_{\text{OUT}}$ satisfying a given validity condition. We assume that V_{IN} is at most countably infinite. In the full version of our paper [9, Section 7], we explain why such an assumption is actually necessary to get any relevant result. In the following, we present each property that a BA protocol needs to satisfy. The first property is *termination*, which may be deterministic (for synchronous protocols) or probabilistic while the second is *agreement*. We define them below:

- **(Termination)** Every honest party decides on an output v_{OUT}.
- **(Probabilistic Termination)** As time goes to infinity, the probability that an honest party has not yet decided on an output value v_{OUT} tends to 0.
- **(Agreement)** If two honest parties output v_{OUT} and v'_{OUT}, then $v_{\text{OUT}} = v'_{\text{OUT}}$.

Before describing the validity property, we need to define *input configurations*. Regardless of the nature of the adversary, these are defined by looking at the honest parties' inputs *after* the adversary has decided which parties to corrupt. Hence, in impossibility results, where we consider a static adversary, input configurations are defined before the protocol's execution. An *input configuration* is a set $I \subseteq \mathcal{P} \times V_{\text{IN}}$ consisting of pairs of honest parties with their inputs: if $(v, P) \in I$, then P is an honest party with input v. Naturally, no party occurs twice in I (i.e., honest parties cannot simultaneously have two inputs). We use the notation $\text{PARTIES}(I)$ to refer to the set of (honest) parties in the input configuration I. Note that, if $P \notin \text{PARTIES}(I)$, then P is corrupted in the input configuration I. Let $\mathcal{I} = \{\text{input configurations } I \subseteq \mathcal{P} \times V_{\text{IN}} \text{ such that } \left|I\right| \geq n - t_s\}$ denote the set of all possible input configurations. We also note the inclusion relation for input configurations: for $I, J \in \mathcal{I}$, $J \subseteq I$ if and only if $\text{PARTIES}(J) \subseteq \text{PARTIES}(I)$ and the parties in $\text{PARTIES}(J)$ have the same input value in both I and J. We say that an input configuration I is maximal if $\text{PARTIES}(I) = \mathcal{P}$. Moreover, as V_{IN} is at most countably infinite, the size of $\mathcal{I}$ is at most countably infinite. A validity property is then defined by a mapping $\text{VAL} : \mathcal{I} \to 2^{V_{\text{OUT}}}$ from honest parties' inputs to *valid* outputs:

- **(Validity)** If $I \in \mathcal{I}$ is the input configuration defined by the honest parties and their inputs, then no honest party outputs $v_{\text{OUT}} \notin \text{VAL}(I)$.

This validity definition matches the one used in [6, 7]. We say that a validity property VAL is *trivial* if $\bigcap_{I \in \mathcal{I}} \text{VAL}(I) \neq \varnothing$. Note that, if this condition holds, we can achieve validity, agreement, and termination with no communication: parties output a value in $\bigcap_{I \in \mathcal{I}} \text{VAL}(I)$.

A validity property VAL is *solvable* if there is a BA protocol *solving* VAL, as defined below.

▶ **Definition 1.** *A protocol* Π *is a* (t_s, t_a)*-secure BA protocol solving a validity property* VAL *if it achieves probabilistic termination, agreement, and validity for the given property* VAL *even when up to* t_s *parties are corrupted if it runs in a synchronous network, and even when up to* t_a *parties are corrupted if it runs in an asynchronous network.*

▶ **Definition 2.** *A protocol* Π *is a* t_s*-secure BA protocol solving a validity property* VAL *if, when running in a synchronous network, it achieves (probabilistic) termination, agreement, and validity for the given property* VAL *even when up to* t_s *parties are corrupted.*

One might be tempted to believe that a $(t_s, 0)$-secure protocol is simply a t_s-secure protocol, but the difference is rather subtle. Namely, a $(t_s, 0)$-secure *network-agnostic* BA protocol also provides guarantees in an asynchronous network if all parties are honest. On the other hand, a t_s-secure (synchronous) BA protocol is not required to provide any guarantees if the synchrony assumptions fail, even if there are no corruptions. We add that this subtle difference would not apply to a purely asynchronous variant of the definition, which is equivalent to (t_a, t_a)-secure *network-agnostic* BA protocol.

Randomness. Our work covers BA protocols which can run in the asynchronous setting. As a result of FLP [19], the protocol considered must be randomized. We consider the randomness as a black box where, at each instant, a party can ask for one or multiple random bits, each set to 0 or 1 uniformly and independently. So the randomness from a party's point of view can be seen as an infinite bitstring being progressively read. There may also be shared randomness, which gives the same result to each party. Therefore, when running a protocol, we can consider its behavior over a probabilistic space $(\Omega, \mathcal{F}, \mu)$ where $\Omega = \left(\{0,1\}^{\mathbb{N}}\right)^k$ for some $k > 0$ is the set of all possible random bits parties will read. $\mathcal{F}$ is the σ-algebra generated by taking all possible prefixes from each bitstring, and μ is the resulting probability measure from having each bit following independently a Bernoulli random variable $\mathcal{B}(0.5)$. From this point on, when mentioning probabilities, like almost surely properties, we refer to the probabilistic space given above.

Executions. For a protocol Π, we define an *execution* ε to be a particular feasible run of Π. In particular, ε contains the input configuration $I \in \mathcal{I}$ from which the protocol started, the behavior of the byzantine parties, and the scheduler's behavior (including whether the network was synchronous or asynchronous). Because an execution depends on the randomness, it is actually a random variable $\varepsilon(\omega)$. We then say that an execution *decides* if all honest parties in the execution eventually decide an output. We note that, given a protocol satisfying probabilistic termination, including the scheduler and strategy of the adversary, an execution for this protocol decides almost surely. We will also be using the term *canonical* to refer to executions occurring in a synchronous network where all messages are delivered exactly Δ units of time after being sent and where all corrupted parties crash right at the beginning of the protocol (i.e., they do not send any messages). We add that, for any given input configuration, there is a unique canonical execution (which, recall, is a random variable).

For a given protocol Π and randomness $\omega \in \Omega$, we say that two executions $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ *cannot be distinguished* by a party P that is honest in both executions if it has the same initial state in $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ (i.e., input and randomness), and receives in both $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ the same messages at the same times. As a consequence, if P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$, then P is in exactly the same state at any time T in $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$. Hence, if P decides a value v_{OUT} at time T in one of the two executions, then it also decides v_{OUT} at time T in the other. We also say that an execution $\varepsilon(\omega)$ is deciding if all honest parties eventually decide when running with randomness ω. If a protocol satisfies probabilistic termination, then an execution is almost surely deciding.

Validity in indistinguishable executions. Goren et al. [25] offer a framework to formalize indistinguishability for randomized protocols. We instead decided to take a different approach by looking at deterministic indistinguishability after fixing the randomness $\omega \in \Omega$. This can allow for easier results as we do not have to consider the whole probabilistic space at a time. For example, some of our proofs require us to define multiple executions depending on ω and cannot be achieved with the framework above. The drawback is that all the assumptions made must hold *almost surely* for the proofs to be correct. Indistinguishability is a powerful tool, especially when considering scenarios where byzantine parties follow the protocol correctly, but with inputs of their own choice. This leads us to the following lemma. The proof is included in Appendix A.

▶ **Lemma 3.** *Let* VAL *be a validity property and* Π *be a* (t_s, t_a)*-resilient* BA *protocol solving* VAL*. Consider two input configurations* I, J *such that* $J \subseteq I$*. Then, the value agreed upon in any execution of* Π *which decides and where the input configuration is* I *must be in* $\text{VAL}(J)$*.*

Intuitively, Lemma 3 ensures that honest parties cannot distinguish between a scenario where all parties in $\text{PARTIES}(I)$ are honest, and one where the parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are, in fact, byzantine. With the lemma in mind, for any validity property VAL, we can define a new validity property VAL′ such that $\text{VAL}'(I) = \bigcap_{J \subseteq I} \text{VAL}(J)$ for all $I \in \mathcal{I}$. Property VAL′ is simultaneously a stronger version of VAL, in that for all $I \in \mathcal{I}$ we have $\text{VAL}'(I) \subseteq \text{VAL}(I)$, but it also has the property of being monotonically *decreasing*, in that for $J \subseteq I$ we have $\text{VAL}'(I) \subseteq \text{VAL}'(J)$. Armed as such, the previous lemma has the following immediate corollaries:

▶ **Corollary 4.** *A solvable validity property* VAL *is trivial if and only if it permits deciding the same value for all maximal input configurations, i.e.,* $\bigcap_{I \in \mathcal{I}, \text{PARTIES}(I) = \mathcal{P}} \text{VAL}(I) \neq \varnothing$*.*

We end by stating a technical lemma that will be of use in the proofs presented in the subsequent sections. The proof of Lemma 5 is included in Appendix A.

▶ **Lemma 5.** *Let* VAL *be a solvable validity property and* Π *a protocol solving* VAL*. Let* $I_1 \in \mathcal{I}$ *be a maximal input configuration. If, for every maximal input configuration* $I_2 \in \mathcal{I}$*, the canonical executions of* Π *for* I_1 *and* I_2 *decide the same value almost surely, then* VAL *is trivial.*

3 Lower Bound on n

In this section, we prove that, for any non-trivial validity property, the condition $n > 2 \cdot t_s + t_a$ is necessary for it to be solvable in the network-agnostic model even if cryptographic setup is available. Our proof will be organized as follows: we start with a preliminary lemma in Section 3.1, which focuses on a simplified setting where $n := 2$. In this setting, at most one

party can crash if the network is synchronous, and both parties are honest if the network is asynchronous. Our preliminary lemma will show a somewhat counter-intuitive result: roughly, the value decided upon in canonical executions is independent of the honest parties' inputs. In Section 3.2, we move from the simplified setting with two parties to a warm-up variant of our main proof. We will be working with n parties, but we will only consider the particular case $t_a := 0$. By reducing this case to our preliminary lemma, we show that the condition $n > 2 \cdot t_s$ is necessary in this setting. Finally, Section 3.3 focuses on the general case, showing that $n > 2 \cdot t_s + t_a$ is necessary by reducing to our preliminary lemma. The main proof will be a more general version of the warm-up proof.

We note that the proofs by Civit et al. [7] rely on reducing any non-trivial validity property to weak validity, enabling lower bounds to focus solely on weak validity. However, their reduction is invalid for randomized protocols and hence cannot be used in our setting.

3.1 Preliminary Lemma

As previously mentioned, our preliminary lemma focuses on a simplified setting with $n := 2$ parties in the network-agnostic model. When the network is synchronous, we allow the adversary to corrupt up to one party ($t_s := 1$). We restrict the adversary's capabilities by only allowing the corrupted party to crash. When the network is asynchronous, both parties are honest ($t_a := 0$). Then, we assume a protocol achieving (probabilistic) termination and agreement in this setting. We show that, almost surely, once randomness is fixed, all canonical executions decide the same value. Note that there may be a set of randomnesses such that some canonical executions do not even decide, but the probability of picking a randomness in this set is zero.

▶ **Lemma 6.** *Assume $n := 2, t_s := 1$, $t_a := 0$, and that corrupted parties are only allowed to crash. Consider a protocol A that achieves probabilistic termination and agreement in this setting. Then, almost surely, there exists a value v such that all canonical executions of A decide v when running.*

Proof. Denote the two parties by P_1 and P_2. Let $\omega \in \Omega$. For every input configuration, we will define a finite number of executions using randomness ω, including the required canonical executions. Under the assumption that all executions defined for ω decide, we show that all canonical executions using randomness ω decide the same value v. From here, our proof proceeds as follows: individually, each defined execution decides almost surely (by construction). The number of executions defined for each input configuration is finite, and the set of input configurations is countable (since V_{IN} is countable). A countable intersection of events holding almost surely holds almost surely. Then, almost surely, all defined executions for ω decide, i.e., our additional assumption holds almost surely, completing the proof.

Hence, from now on, we fix $\omega \in \Omega$ and only consider executions running with this randomness ω. We assume that all executions we consider decide, and we want to show that all canonical executions decide the same value v.

We first consider canonical executions where one of the parties crashes. Let $v_1, v_2 \in V_{\text{IN}}$ be two arbitrary values. First, we consider the canonical execution $\varepsilon_1(\omega)$ of A (in a synchronous network) where P_1 is honest and has input v_1. P_2 is corrupted and crashes at the beginning of the execution. We have assumed that $\varepsilon_1(\omega)$ is deciding, hence P_1 outputs some value v_1' within a finite amount of time $T_1(\omega)$ (note: without receiving any messages from P_2). Second, consider the canonical execution $\varepsilon_2(\omega)$ of A (again, in a synchronous network) where P_2 is honest and has input v_2. P_1 is corrupted and crashes at the beginning of the execution. We

have assumed that $\varepsilon_2(\omega)$ is deciding, hence P_2 outputs some value v_2' within a finite amount of time $T_2(\omega)$. We prove that $v_1' = v_2'$ by defining a third execution $\varepsilon_{1,2}(\omega)$, but, this time, in the asynchronous model. Here, both P_1 and P_2 are honest with inputs v_1 and v_2 respectively. The scheduler delays any message between them until time $\max(T_1(\omega), T_2(\omega))$. This way, P_1 cannot distinguish execution $\varepsilon_{1,2}(\omega)$ from execution $\varepsilon_1(\omega)$, and therefore it outputs v_1' in $\varepsilon_{1,2}(\omega)$ as well. Similarly, P_2 outputs v_2' in $\varepsilon_{1,2}(\omega)$. Recall that A achieves agreement, hence $v_1' = v_2'$. Note that the argument holds for arbitrary $v_1, v_2 \in V_{\text{IN}}$, so we get that *all* canonical executions where one of the parties crashes decide the same value, which we denote by v.

We now consider canonical executions where none of the parties crashes and show that all such executions also decide v. Consider any $v_1, v_2 \in V_{\text{IN}}$ and the canonical execution $\varepsilon_M(\omega)$ of A where both parties are honest and have inputs v_1 and v_2. We also define $\varepsilon_{1,2}(\omega)$ as before. From the previous part, it follows that $\varepsilon_{1,2}(\omega)$ decides v. We will now show that $\varepsilon_M(\omega)$ decides v as well. Based on our assumption, we already know that $\varepsilon_M(\omega)$ decides, so it does so after a finite number of messages ℓ. For the next part of the proof, we first give an intuitive outline. Roughly, we will build a chain of $\ell + 1$ scenarios between execution $\varepsilon_{1,2}(\omega)$ and execution $\varepsilon_M(\omega)$: in each scenario $1 \leq m < \ell$, the network is asynchronous, and the scheduler ensures that the first m messages are received synchronously, while all subsequent messages are delayed sufficiently long. Scenarios 0 and ℓ correspond to executions $\varepsilon_{1,2}(\omega)$ and $\varepsilon_M(\omega)$ respectively, and any two consecutive scenarios will be indistinguishable to the party that sent the last message: this will imply that $\varepsilon_M(\omega)$ also decides v. In the following, we write this idea formally.

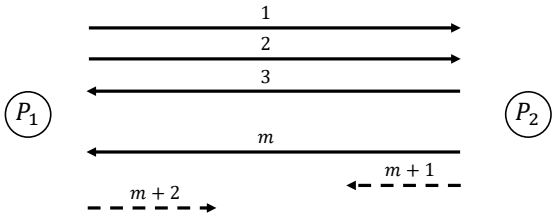


Figure 1 Example of execution $\varepsilon_{M,m}(\omega)$: Any messages sent after the first m messages get delayed until after a value has been decided.

For $0 \leq m \leq \ell$, we consider an execution $\varepsilon_{M,m}(\omega)$ of A in the asynchronous model: both P_1 and P_2 are honest, and they have inputs v_1 and v_2 respectively. In execution $\varepsilon_{M,0}(\omega)$, the scheduler uses the same strategy as that of execution $\varepsilon_{1,2}(\omega)$: the scheduler delays any message in execution $\varepsilon_{M,0}(\omega)$ until time $T = \max(T_1(\omega), T_2(\omega))$. Hence, P_1 and P_2 cannot distinguish between executions $\varepsilon_{M,0}(\omega)$ and $\varepsilon_{1,2}(\omega)$, implying that $\varepsilon_{M,0}(\omega)$ decides v by some time T_0. In execution $\varepsilon_{M,1}(\omega)$, the scheduler allows the first message to be delivered after exactly Δ time. Assume without loss of generality that this message is sent by P_1. All further messages are delayed until time $T_1 > T_0$ (we define the exact time later): hence, P_1 cannot distinguish from this execution and execution $\varepsilon_{M,0}(\omega)$, therefore it outputs v by time T_0, as in execution $\varepsilon_{M,0}(\omega)$. P_2, on the other hand, can distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_{M,0}(\omega)$: however, it cannot distinguish between $\varepsilon_{M,1}(\omega)$ and an execution where the network is, in fact, synchronous, and P_1 has crashed. If P_1 has crashed, P_2 has to output eventually, hence by time T_1'. Hence, the scheduler makes sure to delay all messages until time $T_1 > \max(T_0, T_1')$. Consequently, P_2 has to output v in $\varepsilon_{M,1}(\omega)$ by time T_1. The next executions are defined similarly: in execution $\varepsilon_{M,m}(\omega)$ with $m > 0$, the scheduler allows the

first m messages to be delivered after exactly Δ units of time. The remaining messages are delivered sufficiently late to ensure that the last message's sender is unable to distinguish between $\varepsilon_{M,m}(\omega)$ and execution $\varepsilon_{M,m-1}(\omega)$. Thus, the last message's sender outputs v in execution $\varepsilon_{M,m}(\omega)$, which forces the other honest party to also output v.

We remark that executions $\varepsilon_{M,\ell}(\omega)$ and $\varepsilon_M(\omega)$ are indistinguishable, so $\varepsilon_M(\omega)$ will decide v. Since this holds for arbitrary values v_1, v_2, we have obtained that all canonical executions where no party crashes also output v, hence completing the proof. ◀

3.2 Warm-up: $t_a := 0$

As a warm-up towards the main result, we focus on the particular case $t_a := 0$, and show that the condition $n > 2 \cdot t_s$ is necessary. Intuitively, when $n \leq 2 \cdot t_s$, one cannot distinguish between a scenario where half of the parties crash in a synchronous network, and a scenario where half of the parties are honest but delayed in an asynchronous one. This way, the presence of the asynchronous case, even with no corruptions, allows two disjoint sets of honest parties to only run the protocol within their own set. Since the two sets run the protocol independently, the honest parties agree on a valid value only if the given validity property is trivial. Note that this is the case even for weak validity, which can be solved in the synchronous model for up to $t_s < n$ corruptions. Our result implies that expecting a synchronous protocol to provide guarantees when it runs in a corruption-free asynchronous network impacts the overall resilience.

▶ **Theorem 7.** *Assume $t_s > 0$ and consider a validity property* VAL*. If $n = 2 \cdot t_s$ and there is a $(t_s, 0)$-secure BA protocol solving* VAL*, then* VAL *is trivial.*

Proof. Consider a (possibly randomized) $(t_s, 0)$-secure BA protocol Π that solves VAL when $n = 2 \cdot t_s$. Let I_1 and I_2 be two arbitrary maximal input configurations. We show that the canonical executions of I_1 and I_2 almost surely decide on the same output when run using the same randomness. Then, Lemma 5 ensures that VAL is trivial.

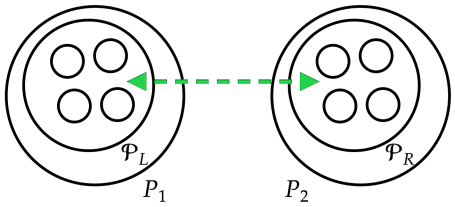


Figure 2 P_1 simulates all parties of $\mathcal{P}_L$ while P_2 simulates all parties of $\mathcal{P}_R$. All the parties of $\mathcal{P}$ communicate between one another not knowing they are being simulated.

We use Π to build a 2-party protocol A that matches the setting described in Lemma 6. For protocol A, we consider the input set $\{0, 1\}$ and output set V_{OUT}, and we denote the two parties running A by P_1 and P_2. Since $n = 2 \cdot t_s$, we may partition the set of n parties $\mathcal{P}$ into two sets $\mathcal{P}_L$ and $\mathcal{P}_R$ of t_s parties each.

Then, as shown in Figure 2, P_1 will simulate the parties in $\mathcal{P}_L$, while P_2 will simulate the parties in $\mathcal{P}_R$. Concretely, in protocol A, P_1 proceeds as follows:

- P_1 simulates all t_s parties of $\mathcal{P}_L$ running Π.
- If P_1 has input 0, then the simulated parties in $\mathcal{P}_L$ use their inputs from I_1. Otherwise, they use their inputs from I_2.

- Messages between the simulated parties in $\mathcal{P}_L$ are received after exactly Δ units of time, and P_1 forwards to P_2 every message sent by a simulated party in $\mathcal{P}_L$ to a party in $\mathcal{P}_R$. Once P_2 receives this message, it immediately forwards it to the simulated receiver in $\mathcal{P}_R$.
- As soon as a party in $\mathcal{P}_L$ decides a value, P_1 decides this value. P_1 continues forwarding messages as described above.

P_2 proceeds identically to P_1, switching $\mathcal{P}_L$ and $\mathcal{P}_R$.

Note that running A in an asynchronous network where both P_1 and P_2 are honest corresponds to running Π in an asynchronous network where all n parties are honest. In addition, running A in a synchronous network where at most one party may crash corresponds to running Π in a synchronous network where at most t_s parties may crash. Then, since Π is a $(t_s, 0)$-secure BA protocol, A achieves probabilistic termination and agreement in the setting described in Lemma 6. Then, applying Lemma 6, we obtain that: almost surely, there is a value v such that all canonical executions of A decide on the same value v. Moreover, the canonical execution of A with input values $(0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Using Lemma 5, we can therefore conclude that VAL is trivial. ◀

3.3 General Result

We are now ready to prove the final statement of this section, presented below.

▶ **Theorem 8.** *Consider a validity property* VAL*, and t_s, t_a such that $t_s > 0$ and $t_s \geq t_a$. If there is a (t_s, t_a)-secure BA protocol solving* VAL *when $n \leq 2 \cdot t_s + t_a$, then* VAL *is trivial.*

Proof. Assume $n = 2 \cdot t_s + t_a$ and that there is a (t_s, t_a)-secure BA protocol Π solving VAL. We consider two input configurations I_1 and I_2 such that $\text{PARTIES}(I_1) = \text{PARTIES}(I_2) = \mathcal{P}$ (i.e., all parties are honest). We show that, almost surely, the canonical executions of I_1 and I_2 decide on the same output. Then, Lemma 5 ensures that VAL is trivial. Similarly to the proof of Theorem 7, we use Π to build a two-party protocol A that matches the setting of Lemma 6. For A, we consider the input space $\{0, 1\}$ and the output space V_{OUT}.

This time, we partition $\mathcal{P}$ into three sets $\mathcal{P}_L$, $\mathcal{P}_M$ and $\mathcal{P}_R$ such that $|\mathcal{P}_L| = |\mathcal{P}_R| = t_s$ and $|\mathcal{P}_M| = t_a$. A crucial difference from the proof of Theorem 7 is that, as shown in Figure 3, each party in $\mathcal{P}$ simulates its own copy of the parties in $\mathcal{P}_M$. Our construction will ensure that, when A runs in the synchronous setting, the two simulated copies of each party in $\mathcal{P}_M$ will be in the exact same state at any point, maintaining the guarantees of Π when at most t_s of the parties are corrupted. Meanwhile, in the asynchronous setting, the copies of the t_a parties in $\mathcal{P}_M$ may be in different states, but Π is able to tolerate t_a byzantine parties in this case. Concretely, in protocol A, party P_1 proceeds as follows:

- P_1 simulates all $t_s + t_a$ parties of $\mathcal{P}_L \cup \mathcal{P}_M$ running Π.
- If P_1 has input 0, then parties in $\mathcal{P}_L \cup \mathcal{P}_M$ take their input from I_1. Otherwise, they take their input from I_2.
- As depicted in Figure 4, the messages sent between the parties simulated by P_1 are exchanged as if all $t_s + t_a$ parties are running independently: each such message is delivered after exactly Δ units of time. Additionally, once a simulated party in $\mathcal{P}_L$ sends a message to a simulated party in $\mathcal{P}_M$, P_1 immediately forwards this message to P_2 as well. Once P_2 receives this message, it immediately forwards it to its own simulated receiver in $\mathcal{P}_M$. The message delay here depends on the type of network that A is running in.
- Messages from a party in $\mathcal{P}_L$ to $\mathcal{P}_R$ are sent from P_1 to P_2. P_2 then forwards each such message to its simulated receiver in $\mathcal{P}_R$.

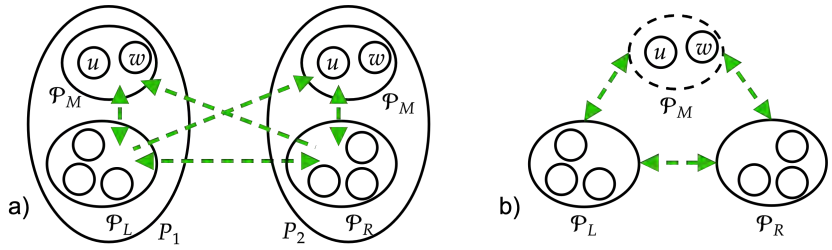


Figure 3 a) P_1 simulates all parties of $\mathcal{P}_L$ and its own copies of the parties in $\mathcal{P}_M$. P_2 simulates all parties of $\mathcal{P}_R$, and its own copy of the parties $\mathcal{P}_M$. The arrows show which simulated group of parties can send messages to which other group.
b) This is how the network looks like from the point of view of a party in $\mathcal{P}_R$ or $\mathcal{P}_M$: they are unaware of the second set $\mathcal{P}_M$ being simulated in parallel.

- P_1 discards any messages sent by the simulated parties in $\mathcal{P}_M$ to parties in $\mathcal{P}_R$.
- As soon as a party in $\mathcal{P}_L$ (but not $\mathcal{P}_M$) decides a value, P_1 decides this value. P_1 continues forwarding messages as described above until all its simulated parties terminate.

The behavior of P_2 is the same as the behavior of P_1, switching $\mathcal{P}_L$ and $\mathcal{P}_R$.

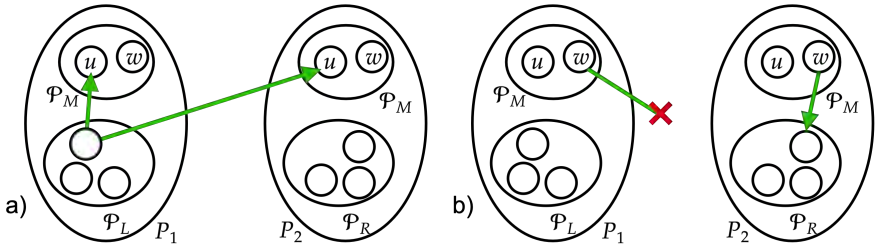


Figure 4 Examples of how different messages are handled.
a) If a simulated party in $\mathcal{P}_L$ wants to send a message to a party in $\mathcal{P}_M$, then two identical messages are actually sent: the first one to the copy of the receiver simulated by P_1, and the second to the copy simulated by in P_2.
b) If a party in $\mathcal{P}_M$ wants to send a message to a party in $\mathcal{P}_R$, if the two parties are simulated by the same entity (here P_2), then the message gets received as expected. Otherwise, the message is discarded and the party in $\mathcal{P}_R$ never receives it.

We now need to analyze A in the setting described by Lemma 6. Running A in an asynchronous network where both parties are honest corresponds to running Π in a network where the communication between parties in $\mathcal{P}_L$ and $\mathcal{P}_R$ is asynchronous. While the simulated copies in $\mathcal{P}_M$ are not necessarily consistent, Π is resilient against $t_a = |\mathcal{P}_M|$ byzantine corruptions, and therefore maintains its guarantees. Hence, A achieves agreement and probabilistic termination in this setting. It remains to show that A also achieves these guarantees in a synchronous network where one of the two parties may crash. Settings where both P_1 and P_2 are honest correspond to running Π in a synchronous network where all n parties are honest. We note that, in this case, for each party in $\mathcal{P}_M$, the copy simulated by P_1 and the copy simulated by P_2 maintain the same state at all times. Settings where at most one of P_1 and P_2 may crash correspond to running Π in a synchronous setting where the t_s parties meant to be simulated only by the party crashing in A are corrupted. As Π tolerates t_s corruptions, it maintains its guarantees. Hence, A achieves agreement and probabilistic termination in this setting as well.

We conclude the proof in an identical manner to the proof of Theorem 7. Applying Lemma 6, we obtain that, almost surely, all canonical executions of A decide on the same value. Moreover, the canonical execution of A with input values $(0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Using Lemma 5, we can therefore conclude that VAL is trivial. ◀

4 Lower Bound on n Without Cryptographic Setup

The previous section has proven that the condition $n > 2 \cdot t_s + t_a$ is necessary regardless of whether a cryptographic setup is available or not. We now focus on settings without cryptographic setup and prove an even stronger condition. We show that, in such settings, the condition $n > 3 \cdot t_s$ is necessary even in the synchronous model. Since a protocol achieving (t_s, t_a)-secure BA in the network-agnostic model also achieves t_s-secure BA in the synchronous model, this bound immediately applies to the network-agnostic model. We add that our result extends the requirement of $n > 3 \cdot t_s$ provided by Civit et al. [7] for synchronous deterministic protocols to cover randomized protocols as well.

4.1 Preliminary Lemma

Similarly to the outline of Section 3, we first consider a setting with three parties, out of which at most one is byzantine. Afterwards, we focus on the general case. Lemma 9 is an improvement over the result of Fischer, Lynch, and Merritt for weak validity [18]: the result of [18] only applies for weak validity and protocols must always decide in a finite amount of time, which is a lot stronger than probabilistic termination. Our proof uses the main core idea but improves it to lift these restrictions. Roughly, we will be running A in a larger ring containing multiple copies of each party, as depicted in Figure 5.

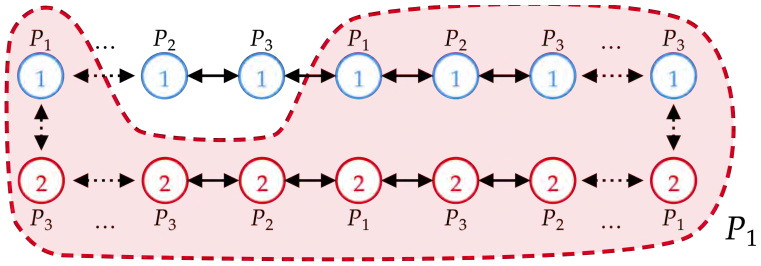


Figure 5 Defining the behavior of a byzantine party (here P_1).

The ring is constructed from two canonical executions with different input configurations. Parties adjacent in this ring cannot distinguish between the ring and the original setting of three parties, as the third party may be byzantine and simulate the rest of the ring. This forces parties adjacent in the ring to output the same value, which implies that the two original executions lead to the same output. We defer the formal proof to Appendix B.1.

▶ **Lemma 9.** *Consider $n := 3$ parties in a synchronous network, and assume a protocol A that achieves (probabilistic) termination and agreement in this setting even when up to one party is byzantine. Then, almost surely, all canonical executions of A decide the same value.*

4.2 General Result

To prove our main result in this setting, we use a strategy similar to the proof of Theorem 7. Concretely, we assume an n-party protocol that achieves VAL whenever $n \geq 3 \cdot t_s$ and use it to construct a three-party protocol that contradicts Lemma 9. We defer the formal proof to Appendix B.2.

▶ **Theorem 10.** *Assume $t_s > 0$ and consider a validity property VAL. If there is a t_s-secure BA protocol solving VAL in the synchronous model when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then VAL is trivial.*

Then, as any lower bound in the synchronous model also holds in the network-agnostic model, Theorem 10 immediately implies the following corollary.

▶ **Corollary 11.** *Consider a validity property VAL, and let t_s, t_a such that $t_s > 0$ and $t_s \geq t_a$. If there is a (t_s, t_a)-secure BA protocol solving VAL when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then VAL is trivial.*

5 Similarity Condition

The lower bounds on n presented in the previous sections are indeed necessary, but not yet sufficient. We may instantiate, for instance, the input space as the (finite) set of vertices of a (publicly known) labeled graph with maximum clique size $\omega \geq 3$. We consider convex validity, under the so-called *monophonic convexity*. For this example, network-agnostic BA requires the stronger lower bound $n > \max(\omega \cdot t_s, \omega \cdot t_a + t_s, 2 \cdot t_s + t_a)$ [11]. Roughly, if any of the conditions $n > \omega \cdot t_s$ and $n > \omega \cdot t_a + t_s$ fails to hold, one can find scenarios where the simple presence of byzantine parties (following the protocol correctly, with inputs of their choice) prevents the honest parties from obtaining a valid output. In this section, we prove the need of one more condition that captures these validity-dependent requirements, and enables the honest parties to find a valid output even if their view over the honest inputs is not accurate. This additional condition matches the *similarity* condition of [8] for the partially synchronous model, and the *containment* condition of [7] in the synchronous model.

We need to establish a few notions. The first is the notion of *neighbors* of an input configuration. The neighbors of I, denoted by $\text{NEIGHBORS}(I)$, are the input configurations J such that the parties in $\text{PARTIES}(I) \cap \text{PARTIES}(J)$ hold the same input values in I and J. Formally, $\text{NEIGHBORS}(I) := \{J \in \mathcal{I} : \ \forall P \in \mathcal{P}, \text{ if } (v_1, P) \in I \text{ and } (v_2, P) \in J \text{ then } v_1 = v_2\}$. The definition of neighbors is symmetric (if $J \in \text{NEIGHBORS}(I)$ then $I \in \text{NEIGHBORS}(J)$).

The second notion is that of *similar configurations* of an input configuration I, denoted by $\text{SIMILAR}(I)$. These are input configurations that may be indistinguishable from I. In the synchronous model, these are configurations $J \in \text{NEIGHBORS}(I)$ such that $J \subseteq I$. Roughly, this models scenarios where some of the parties are corrupted, but follow the protocol correctly with inputs of their own choice. In the asynchronous model, these are configurations $J \in \text{NEIGHBORS}(I)$ containing $n - t_a$ parties: this additionally takes into account that at most t_a honest parties may be isolated due to network delays. Hence, we define $\text{SIMILAR}(I)$ as $\text{SIMILAR}(I) = \{J \in \text{NEIGHBORS}(I) : J \subseteq I\} \bigcup \{J \in \text{NEIGHBORS}(I) : |J| \geq n - t_a\}$.

We may now define the similarity condition, which Lemma 13 proves to be necessary for the network-agnostic model. Lemma 13 adapts Theorem 2 of [8] and Lemma 8 of [7] to the network-agnostic model, and it relies on standard indistinguishability arguments. We defer the proof to Appendix C.

▶ **Definition 12** (Similarity condition). *We say that a validity property* VAL *satisfies the* similarity condition *if there is a Turing-computable function* $\sigma : \mathcal{I} \mapsto V_O$ *such that, for any input configuration* $I \in \mathcal{I}$, $\sigma(I) \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$.

Note that the existence of such a function σ also implies $\forall I \in \mathcal{I}, \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J) \neq \varnothing$.

▶ **Lemma 13.** *If a validity property* VAL *is solvable in the network-agnostic model, then* VAL *satisfies the similarity condition.*

It is worth noting that, in the proof of this lemma, the deterministic Turing function σ is defined based on a randomized protocol that solves VAL. This is justified because, as discussed in the proof, when time complexity is not taken into account, probabilistic Turing machines are as expressive as deterministic ones.

6 Sufficiency and Main Result

We now show that the conditions presented in the previous sections are not only necessary, but also sufficient, hence proving our main result, stated below.

▶ **Theorem 14.** *Assume a non-trivial validity condition* VAL*. Then, there is a* (t_s, t_a)*-secure protocol solving* VAL *if and only if the following conditions hold:*

- VAL *satisfies the similarity condition.*
- $n > 3 \cdot t_s$ *or, if PKI is available,* $n > 2 \cdot t_s + t_a$.

The remainder of the section describes a protocol that matches our necessary conditions, as stated in the lemma below. Theorem 14 then follows from combining Lemma 15 with the requirements discussed previously: the lower bounds on n, described in Theorem 8 and Corollary 11, plus the similarity condition, proven to be necessary in Lemma 13.

▶ **Lemma 15.** *Assume a validity condition* VAL *that satisfies the similarity condition. Then, if* $n > 3 \cdot t_s$*, or, assuming that PKI is available, if* $n > 2 \cdot t_s + t_a$*, there is a* (t_s, t_a)*-secure* BA *protocol solving* VAL*.*

Our construction behind Lemma 15 generalizes the network-agnostic BA protocol of [11] that solves convex validity. The parties distribute their input values using a protocol achieving *Agreement on a Core-Set* (ACS), which provides them with an identical view over the inputs, i.e., with a potential input configuration. This is a (randomized) communication primitive enabling identical views in the pure asynchronous model [2,3]. We make use of the ACS definition of [11], included below, which differs from the standard definition by providing stronger properties in the synchronous model: it additionally ensures that all honest inputs are included in the common view if the network is synchronous. This property is essential for matching the higher resilience threshold t_s in a synchronous network. This way, the parties can apply a deterministic decision over the view agreed upon in ACS and obtain an output.

▶ **Definition 16.** *Let* Π *be a protocol where every party* P_i *holds an input* v_i *and outputs a set* $I_i \in \mathcal{I}$ *consisting of at least* $n - t_s$ *value-sender pairs. We consider the following properties in addition to those presented in Section 2:*

- ***Integrity****: If* P_i *and* P_j *are both honest and* $P_i \in \text{PARTIES}(I_j)$*, then* $(P_i, v_i) \in I_j$.
- ***Honest Core****: If an honest party outputs* I*, then* PARTIES(I) *contains all honest parties.*

Then, we say that Π *is a* (t_s, t_a)*-secure* ACS *protocol if it achieves the following:*

- *Probabilistic termination, agreement, integrity, and honest core when running in a synchronous network where up to* t_s *parties are corrupted;*

- *Probabilistic termination, agreement, and integrity when running in an asynchronous network where up to t_a parties are corrupted.*

We make use of the ACS construction of [11], described by the result below.

▶ **Theorem 17** ([11]). *Consider n, t_s, t_a such that $t_a \leq t_s$. If $3 \cdot t_s < n$, or, if PKI is available and $2 \cdot t_s + t_a < n$, there is a (t_s, t_a)-secure ACS protocol Π_{ACS}.*

We may now present the proof of Lemma 15, describing our protocol.

Proof of Lemma 15. Our BA protocol solving VAL proceeds as follows: the parties distribute their input values using the protocol Π_{ACS} described in Theorem 17. Π_{ACS} provides the parties with the same output set $I \in \mathcal{I}$ of at least $n - t_s$ value-sender pairs, representing a potential input configuration. Once the parties obtain this set, they output $\sigma(I)$, where σ is the Turing-computable function provided by VAL satisfying the similarity condition.

(t_s, t_a)-secure BA solving VAL

Code for party P_i with input v_i

1: Join Π_{ACS} with input v_i and obtain the output set I.
2: Output $\sigma(I)$ and terminate.

Regardless of whether the network is synchronous or asynchronous, since Π_{ACS} achieves probabilistic termination, our BA protocol achieves probabilistic termination as well. In addition, since Π_{ACS} achieves agreement, the parties compute their output identically, and therefore our BA protocol achieves agreement.

It remains to prove that the honest parties' output is in $\text{VAL}(H)$, where H denotes the (actual) input configuration (containing only the honest parties and their inputs). Since $\sigma(I) \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$, it will be sufficient to show that $H \in \text{SIMILAR}(I)$. Regardless of the type of network, the integrity property of Π_{ACS} ensures that $H \in \text{NEIGHBORS}(I)$. If the network is asynchronous, $\left|H\right| \geq n - t_a$ and therefore $H \in \text{SIMILAR}(I)$. Otherwise, if the network is synchronous, the honest core property of Π_{ACS} ensures that honest parties and their inputs are included in I. Then, $H \subseteq I$ and therefore $H \in \text{SIMILAR}(I)$ in this case as well. Thus, as $\sigma(I) \in \bigcap_{H \in \text{SIMILAR}(I)} \text{VAL}(H)$, we get that the value agreed upon is in $\text{VAL}(H)$, which proves the validity of the protocol and concludes the proof. ◀

7 Conclusions and Future Work

We investigated the conditions that a validity property needs to satisfy in order to be solvable in the network-agnostic model and established the necessary and sufficient conditions. Our results demonstrate that solving a non-trivial validity property VAL requires **(i)** that VAL satisfies the *similarity condition*, and **(ii)** that $n > 2 \cdot t_s + t_a$ assuming a public key infrastructure, or $n > 3 \cdot t_s$ otherwise. Further, we provided a universal protocol that solves a given validity property whenever these established conditions are met. Our characterization follows the line of works of [7, 8] focusing on the partially synchronous model and on the synchronous model. At the same time, it generalizes prior results on when network-agnostic BA can be achieved [4, 11] from fixed validity properties to arbitrary validity properties.

While our work provides a complete answer for solvability, we leave a number of exciting problems open. Our results and approach do not hold if we allow the protocol to fail with some probability $\varepsilon > 0$. Focusing on this weaker setting would likely allow more efficient protocols. Future works could also extend our characterization to cover settings where both

V_{IN} and V_{OUT} are uncountable sets and consider proving message complexity lower bounds. Other promising directions would aim to improve the efficiency of our universal protocol, or to generalize our results further to network-dependent validity properties (that allow weaker guarantees if the network is asynchronous) [10], or to weaker agreement definitions [11].

References

1 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Principles of Distributed Systems*, pages 229–239, Berlin, Heidelberg, 2005. doi:10.1007/11516798_17.

2 Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 52–61, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167109.

3 Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 183–192, New York, NY, USA, 1994. Association for Computing Machinery. doi:10.1145/197917.198088.

4 Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference*, pages 131–150. Springer, 2019. doi:10.1007/978-3-030-36030-6_6.

5 Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal byzantine-resilient convergence in uni-dimensional robot networks. *Theoretical Computer Science*, 411(34):3154–3168, 2010. doi:10.1016/j.tcs.2010.05.006.

6 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Matteo Monti, and Manuel Vidigueira. Every Bit Counts in Consensus. In *37th International Symposium on Distributed Computing (DISC 2023)*, volume 281, pages 13:1–13:26, Dagstuhl, Germany, 2023. doi:10.4230/LIPIcs.DISC.2023.13.

7 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. All byzantine agreement problems are expensive. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 157–169, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3662158.3662780.

8 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. On the validity of consensus. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 332–343, 2023. doi:10.1145/3583668.3594567.

9 Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer. Validity in network-agnostic byzantine agreement. Full version of this paper. doi:10.48550/arXiv.2410.19721.

10 Andrei Constantinescu, Diana Ghinea, Lioba Heimbach, Zilin Wang, and Roger Wattenhofer. A Fair and Resilient Decentralized Clock Network for Transaction Ordering. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, volume 286, pages 8:1–8:20, Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.OPODIS.2023.8.

11 Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex Consensus with Asynchronous Fallback. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319, pages 15:1–15:23, Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.DISC.2024.15.

12 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In *Theory of Cryptography Conference*, pages 623–653. Springer, 2021. doi:10.1007/978-3-030-90459-3_21.

13 Giovanni Deligios and Mose Mizrahi Erbes. Closing the efficiency gap between synchronous and network-agnostic consensus. In *Advances in Cryptology – EUROCRYPT 2024*, pages 432–461, 2024. doi:10.1007/978-3-031-58740-5_15.

14 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, May 1986. doi:10.1145/5925.5931.

15 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985. doi:10.1145/2455.214112.

16 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

17 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988. doi:10.1145/42282.42283.

18 Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In Michael A. Malcolm and H. Raymond Strong, editors, *4th ACM PODC*, pages 59–70. ACM, August 1985. doi:10.1145/323596.323602.

19 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

20 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, pages 70–80, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538442.

21 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Multidimensional approximate agreement with asynchronous fallback. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 141–151, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558481.3591105.

22 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Brief Announcement: Communication-Optimal Convex Agreement. In *The 43rd ACM Symposium on Principles of Distributed Computing (PODC), Nantes, France*, June 2024.

23 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Communication-optimal convex agreement. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '25, pages 39–49, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3732772.3733551.

24 John T. Gill. Computational complexity of probabilistic turing machines. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 91–95, 1974. doi:10.1145/800119.803889.

25 Guy Goren, Yoram Moses, and Alexander Spiegelman. Probabilistic indistinguishability and the quality of validity in byzantine agreement. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, pages 111–125, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3558535.3559789.

26 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Byzantine Preferential Voting. In *14th Conference on Web and Internet Economics (WINE), Oxford, United Kingdom*, December 2018. doi:10.1007/978-3-030-04612-5_22.

27 Darya Melnyk and Roger Wattenhofer. Byzantine agreement with interval validity. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 251–260, 2018. doi:10.1109/SRDS.2018.00036.

28 Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015. doi:10.1007/s00446-014-0240-5.

29 Gil Neiger. Distributed consensus revisited. *Information Processing Letters*, 49(4):195–201, 1994. doi:10.1016/0020-0190(94)90011-6.

30 Thomas Nowak and Joel Rybicki. Byzantine Approximate Agreement on Graphs. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146, pages 29:1–29:17, Dagstuhl, Germany, 2019. doi:10.4230/LIPIcs.DISC.2019.29.

31 David Stolz and Roger Wattenhofer. Byzantine Agreement with Median Validity. In *19th International Conference on Principles of Distributed Systems (OPODIS 2015)*, volume 46, pages 1–14, Dagstuhl, Germany, 2016. doi:10.4230/LIPIcs.OPODIS.2015.22.
32 Nitin H. Vaidya and Vijay K. Garg. Byzantine vector consensus in complete graphs. In Panagiota Fatourou and Gadi Taubenfeld, editors, *32nd ACM PODC*, pages 65–73. ACM, July 2013. doi:10.1145/2484239.2484256.

Appendix

A Preliminaries: Missing Proofs

▶ **Lemma 3.** *Let* VAL *be a validity property and* Π *be a* (t_s, t_a)*-resilient* BA *protocol solving* VAL*. Consider two input configurations* I, J *such that* $J \subseteq I$*. Then, the value agreed upon in any execution of* Π *which decides and where the input configuration is* I *must be in* $\text{VAL}(J)$*.*

Proof. Consider a deciding execution ε_I of Π for I. Construct the execution ε_J, which is identical to ε_I except that its input configuration is J instead of I. Observe that ε_J is an execution of Π for J. Indeed, parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ (which are byzantine) can act as if they are honest and behave exactly as they do in ε_I. The honest parties in J cannot distinguish between ε_I and ε_J, so the agreed-upon value in both executions must be the same. Since ε_J is a valid execution of Π for J, this value must be in $\text{VAL}(J)$. ◀

▶ **Lemma 5.** *Let* VAL *be a solvable validity property and* Π *a protocol solving* VAL*. Let* $I_1 \in \mathcal{I}$ *be a maximal input configuration. If, for every maximal input configuration* $I_2 \in \mathcal{I}$*, the canonical executions of* Π *for* I_1 *and* I_2 *decide the same value almost surely, then* VAL *is trivial.*

Proof. Note that a countable intersection of events that happen almost surely happens almost surely. We assume the lemma's condition and want to prove that VAL is trivial. Because V_{IN} is at most countably infinite, the number of maximal input configurations $I_2 \in \mathcal{I}$ is at most countably infinite. By taking an intersection of events, one for each maximal $I_2 \in \mathcal{I}$, that happen almost surely by the lemma condition, we get that, almost surely, for all $I_2 \in \mathcal{I}$ the canonical executions of Π for I_1 and I_2 decide the same value. Hence, because it happens with probability 1, it means we can find $\omega \in \Omega$ such that this event happens. That is, for all maximal $I_2 \in \mathcal{I}$, the canonical executions of Π for I_1 and I_2 with randomness ω decide the same value. This implies that, for all maximal $I \in \mathcal{I}$, the canonical execution of Π on I with randomness ω decides the same value $v \in V_{\text{OUT}}$. Therefore, by Corollary 4, VAL is trivial. ◀

B No Cryptographic Setup: Missing Proofs

B.1 Proof of Lemma 9

We describe the proof of Lemma 9, restated below.

▶ **Lemma 9.** *Consider* $n := 3$ *parties in a synchronous network, and assume a protocol* A *that achieves (probabilistic) termination and agreement in this setting even when up to one party is byzantine. Then, almost surely, all canonical executions of* A *decide the same value.*

To prove our statement, we will be running A in a larger ring containing multiple copies of each party, as depicted in Figure 6. The ring is constructed from two canonical executions with different input configurations. We will show that parties adjacent in this ring cannot

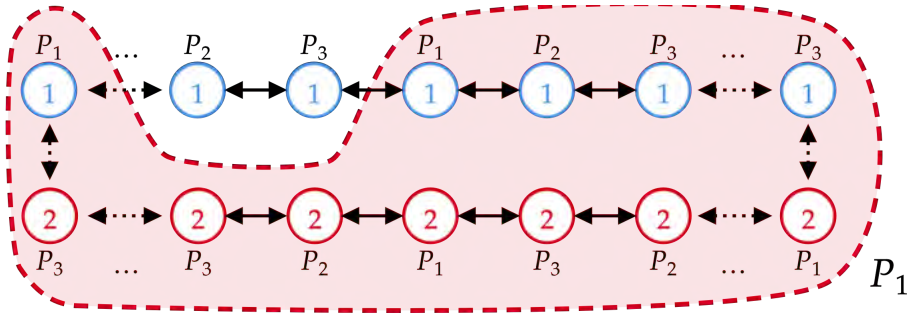


Figure 6 Defining the behavior of a byzantine party (here P_1).

distinguish between the ring and the original setting of three parties, as the third party may be byzantine and simulate the rest of the ring. This will force parties adjacent in the ring to output the same value, which, in turn, guarantees that the two original executions lead to the same output.

Restricting the probabilistic space. In the following proof, we will consider a countable amount of different executions. Since Π satisfies probabilistic termination, it means that each of these executions are deciding almost surely. Because the countable union of almost surely events happens almost surely, this means that the event $E = \{$"all executions considered are deciding"$\}$ happens almost surely. So it is enough to prove that all canonical execution of A in E decide the same value to get the proof.

Constructing the ring. In order to formally describe the construction of the ring, we have to fix two executions with the same randomness. We denote the three parties by $\mathcal{P} = \{P_1, P_2, P_3\}$, and we consider two arbitrary maximal input configurations I_1, I_2. Let $\omega \in E$, and we consider a canonical execution $\varepsilon_1(\omega)$ with input configuration I_1, and a canonical execution $\varepsilon_2(\omega)$ with input configuration I_2. By definition of E, these two executions are deciding.

As $\varepsilon_1(\omega)$ is a deciding execution, there is a number of rounds $r_1(\omega) > 0$ such that, in $\varepsilon_1(\omega)$, all honest parties have decided the same value within $r_1(\omega)$ rounds. Similarly, there is an $r_2(\omega)$ such that, in the canonical execution $\varepsilon_2(\omega)$, all honest parties have decided the same value within $r_2(\omega)$ rounds. Let $r := \max\{r_1(\omega), r_2(\omega)\}$, r implicitly depends on ω.

To construct the ring depicted in Figure 6, we make $4(r+1)$ copies of each party P_i. Out of the $4(r+1)$ copies of party P_i, $2(r+1)$ will be the copies of P_i having its input value from I_1 and $2(r+1)$ will be the copies of P_i with input from I_2. The copies of P_i are then denoted by $P_{k,i,j}$, where $k \in \{1,2\}$, $i \in \{1,2,3\}$, and $j \in \{0,1,\ldots,2r\}$. The copies indexed by $k := 1$ are the ones on the top row of Figure 6, while the copies indexed by $k := 2$ are the ones on the bottom row. We now connect these copies via bidirectional communication channels:

- For $k \in \{1,2\}$ and $j \in \{0,1,\ldots,2r\}$, we add a channel between $P_{k,1,j}$ and $P_{k,2,j}$, and one between $P_{k,2,j}$ and $P_{k,3,j}$.
- Then, to complete the path on the row indicated by index $k := 1$, for every index $j \in \{0,1,\ldots,2r-1\}$, we add a channel between $P_{1,3,j}$ and $P_{1,1,j+1}$.
- Similarly, to complete the path on the row indicated by $k := 2$, for each $j \in \{0,1,\ldots,2r-1\}$, we add a channel between $P_{2,1,j}$ and $P_{2,3,j+1}$.
- We now connect the two rows and hence complete the ring: we add a channel between $P_{1,1,0}$ and $P_{2,3,0}$, and one between $P_{1,3,2r}$ and $P_{2,1,2r}$.

Outputs of adjacent copies. We consider a synchronous execution $\varepsilon(\omega)$ on the ring, where each copy $P_{k,i,j}$ runs A as party P_i, using the input value assigned to it in input configuration I_k. We assume that messages are received exactly Δ units of time after being sent. In the lemma below, we show that adjacent copies in the ring, denoted by Q_1 and Q_2, obtain the same output.

▶ **Lemma 18.** *Consider two parties Q_1 and Q_2 that are adjacent in the ring. Then, in execution $\varepsilon(\omega)$, Q_1 and Q_2 obtain outputs, and they output the same value.*

Proof. Without loss of generality, assume that Q_1 and Q_2 are copies of P_1 and P_2. We consider an execution $\varepsilon'(\omega)$ of A with three parties. In execution $\varepsilon'(\omega)$, P_1 and P_2 have the same input values as Q_1 and Q_2. P_3 is byzantine and simulates the additional parties in the ring, so they have the same behavior as in execution $\varepsilon(\omega)$. All messages are delivered in exactly Δ time, similarly to execution $\varepsilon(\omega)$.

We remark that execution $\varepsilon'(\omega)$ and execution $\varepsilon(\omega)$ are identical. Hence, our execution over the ring is equivalent to running A in a synchronous setting with three parties out of which one is corrupted. Therefore, A maintains its properties, namely probabilistic termination and agreement, on the ring as well. It follows that Q_1 and Q_2 both obtain outputs, and they output the same value. ◀

Outputs on the entire ring. Lemma 18 establishes that adjacent copies in the ring output the same value in execution $\varepsilon'(\omega)$. Using induction, we prove that all parties output the same value v in $\varepsilon(\omega)$. As a consequence, the copies $P_{1,i,r}$ and $P_{2,i,r}$ of each party P_i output v. This will imply that, in the two executions $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ we defined when constructing the ring, all honest parties output the same value.

▶ **Lemma 19.** *In execution $\varepsilon(\omega)$, all parties output, and they output the same value.*

Proof. We prove by induction that, in execution $\varepsilon(\omega)$, after $r' \leq r$ rounds, for $i \in \{1, 2, 3\}$, parties $P_{1,i,j}$ with $j \in \{r - (r - r'), \ldots, r + (r - r')\}$ are in the same state as party P_i in the canonical execution $\varepsilon_1(\omega)$. The base case $r' := 0$ considers the very beginning of the executions $\varepsilon(\omega)$ and $\varepsilon_1(\omega)$, where the parties have not yet received any messages. Their state then is only defined by their input value and ω. Each party $P_{1,i,0}$ takes its input from I_1, which the case for P_i in execution $\varepsilon_1(\omega)$ as well. We thus obtain that parties $P_{1,i,0}$ in the ring have the same input state as party P_i.

For the induction step, assume that our claim holds for $r' < r$, and we prove that it also holds for $r' + 1$. Let $\mathcal{P}_{r'}$ denote the set of parties for which we proved they were in the same state as in $\varepsilon_1(\omega)$ after r' rounds, and let $\mathcal{P}_{r'+1}$ be the set of parties for which we want to prove that the claim holds after $r' + 1$ rounds. The set of direct neighbors of $\mathcal{P}_{r'+1}$ (including themselves) in the ring is $\mathcal{P}_{r'}$. Moreover, within one round, parties are only able to receive messages from their direct neighbors. Using the induction hypothesis, we get that parties in $\mathcal{P}_{r'+1}$ receive exactly the same messages at the $(r' + 1)$-th round in execution $\varepsilon_1(\omega)$ and $\varepsilon(\omega)$, therefore they stay in the same state after round $r' + 1$.

As a consequence, P_i and $P_{1,i,r}$ are in the same state after the r-th round in executions $\varepsilon_1(\omega)$ and $\varepsilon(\omega)$ respectively. However, we have assumed that all parties in $\mathcal{P}$ output by round $r_1 \leq r$ in execution $\varepsilon_1(\omega)$. Therefore, in execution $\varepsilon(\omega)$, $P_{1,i,r}$ outputs the same value v_1 as P_i in $\varepsilon_1(\omega)$. With a symmetric argument, one can show that parties $P_{2,i,r}$ obtain the same output v_2 as P_2 in $\varepsilon_2(\omega)$. This enables us to conclude that $\varepsilon_1(\omega)$ and $\varepsilon_2(\omega)$ decide the same value $v_1 = v_2$ using Lemma 18. ◀

Assumption on deciding execution. Before concluding the proof of Lemma 9 using Lemma 19, we need to discuss the assumptions over deciding executions and make sure that event E happens almost surely. For every possible input configuration of size three, we consider a finite amount of fixed executions. Moreover, we take into account that there is only at most a countably infinite amount of input configurations (because we assumed V_{IN} was countable). Therefore, we consider in total a countable union of a finite amount of executions, which is countable. Therefore, the event that all these executions are deciding E happens almost surely, which concludes the proof of Lemma 9.

B.2 Proof of Theorem 10

We present the proof of Theorem 10, restated below.

▶ **Theorem 10.** *Assume $t_s > 0$ and consider a validity property* VAL*. If there is a t_s-secure* BA *protocol solving* VAL *in the synchronous model when no cryptographic setup is available and $n \leq 3 \cdot t_s$, then* VAL *is trivial.*

Proof. Assume $t_s \geq \lfloor n/3 \rfloor$, and that there is a t_s-secure BA protocol Π solving VAL in the synchronous model.

We consider two arbitrary maximal input configurations I_1 and I_2. We show that all canonical executions of I_1 and I_2 almost surely decide on the same output value. Then, Lemma 5 ensures that VAL is trivial. Similarly to the proof of Theorem 7 and Theorem 8, we use Π to build a protocol A for three parties, denoted by P_1, P_2, P_3, that matches the setting of Lemma 9. For protocol A, we consider the input space $\{0, 1\}$ and the output space V_{OUT}.

To do so, we partition $\mathcal{P}$ into three sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ of size at most t_s each. As shown in Figure 7, in protocol A, P_i simulates all the parties in set $\mathcal{P}_i$. P_i ensures that all messages between the simulated parties in $\mathcal{P}_i$ get delivered within Δ time. In addition, P_i forwards any message sent from a party it simulates to a party in set $\mathcal{P}_j$ $(j \neq i)$ to P_j. When P_j receives this message, it immediately forwards it to the simulated receiver. If P_i has input 0, the simulated parties in $\mathcal{P}_i$ take their input from I_1. Otherwise, they take their input from I_2. When a party in $\mathcal{P}_i$ outputs a value v, P_i outputs v.

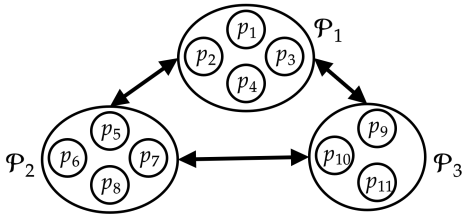


Figure 7 Partitioning $\mathcal{P}$ into 3 sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ with $n = 11$.

Since Π is a t_s-resilient BA protocol when $n \leq 3 \cdot t_s$, we obtain that A achieves probabilistic termination and agreement even when one of the three parties is byzantine. Then, Lemma 9 ensures that, almost surely, all deciding canonical executions of A lead to the same output value.

Moreover, we remark that, by construction, the canonical execution of A with input values $(0, 0, 0)$ matches the canonical execution of I_1 for Π. Similarly, the canonical execution of A with input $(1, 1, 1)$ matches the canonical execution of I_2. As a consequence, canonical executions of I_1 and I_2 decide the same value almost surely. Applying Lemma 5, we conclude that VAL is trivial. ◀

C Similarity Condition: Missing Proof

▶ **Lemma 13.** *If a validity property* VAL *is solvable in the network-agnostic model, then* VAL *satisfies the similarity condition.*

Proof. Assume that Π is a network-agnostic BA protocol solving VAL. Consider an input configuration I.

In the following proof, we will consider a finite amount of different executions. Since Π satisfies probabilistic termination, it means that each of these executions are deciding almost surely. Because the finite union of almost surely events happens almost surely, all of these executions will decide almost surely. So, there exists some $\omega \in \Omega$ which we fix such that all the executions below are deciding when ran with randomness ω.

We first consider the canonical execution $\varepsilon_1(\omega)$ of Π on I. As Π achieves network-agnostic BA, all honest parties output the same value v in execution $\varepsilon_1(\omega)$. We want to prove that $v \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$, hence we show that $v \in \text{VAL}(J)$ for every $J \in \text{SIMILAR}(I)$. Using the definition of $\text{SIMILAR}(I)$, we split the analysis as follows:

(i) $J \in \text{NEIGHBORS}(I)$ such that $J \subseteq I$. Consider an execution $\varepsilon_2(\omega)$ where the input configuration is J and the network is synchronous. Parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are byzantine, but follow the protocol correctly using the values assigned to them in I as inputs. Parties in $\mathcal{P} \setminus \text{PARTIES}(I)$ crash at the start of the execution. Parties in $\text{PARTIES}(J)$ cannot distinguish between $\varepsilon_2(\omega)$ and $\varepsilon_1(\omega)$, so the output value v must also satisfy $v \in \text{VAL}(J)$.

(ii) $J \in \text{NEIGHBORS}(I)$ such that $|J| \geq n - t_a$. First, note that $I \cap J \neq \varnothing$: since $|I| \geq n - t_s$ and $|J| \geq n - t_a$, we obtain that $\left|I \cap J\right| \geq n - t_s - t_a$. As VAL is solvable, Theorem 8 ensures that $n > 2 \cdot t_s + t_a$, and allows us to conclude that $\left|I \cap J\right| > 0$.
Let $P \in I \cap J$. We consider an execution $\varepsilon_3(\omega)$ where the input configuration is J and the network is asynchronous. Similarly to execution $\varepsilon_2(\omega)$, parties in $\text{PARTIES}(I) \setminus \text{PARTIES}(J)$ are byzantine, but follow the protocol correctly using the values assigned to them in I as inputs, and parties in $\mathcal{P} \setminus (\text{PARTIES}(I) \cup \text{PARTIES}(J))$ crash at the very beginning of the execution. All messages are delivered in a synchronous way, except for the messages sent from parties in $\text{PARTIES}(J) \setminus \text{PARTIES}(I)$. These are delayed until after party P outputs: this is possible as P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_3(\omega)$ and therefore it has to obtain an output without receiving these messages. In addition, the fact that P cannot distinguish between $\varepsilon_1(\omega)$ and $\varepsilon_3(\omega)$ ensures that the output v agreed upon in $\varepsilon_1(\omega)$ satisfies $v \in \text{VAL}(J)$.

Therefore, the output v in execution $\varepsilon_1(\omega)$ satisfies $v \in \bigcap_{J \in \text{SIMILAR}(I)} \text{VAL}(J)$. So when running $\varepsilon_1(\varepsilon)$, if this execution terminates (which happens almost surely), then this value can be used for $\sigma(I)$.

We now explain how to get a Turing computable function out of protocol Π and this property. We first remark that Π is a randomized protocol, which can be simulated by a probabilistic Turing machine. However, when time complexity is not taken into account, a probabilistic Turing machine is as expressive as a regular deterministic Turing machine (see Theorem 2 from [24]: for a given number of random bits used, we can try all of them in exponential time). Therefore, we can simulate a deciding execution of ε_1 using a deterministic Turing machine. Because there are no byzantine parties, we can also remove the public-key infrastructure assumption which was used as a black box (for example, one can replace the signing function with one that returns the message concatenated with the id of the party). This gives us a Turing function to compute $\sigma(I)$. ◀

Model-Agnostic Approximation of Constrained Forest Problems

Corinna Coupette
Aalto University, Finland
Max Planck Institute for Informatics, Saarbrücken, Germany

Alipasha Montaseri
Sharif University of Technology, Tehran, Iran

Christoph Lenzen
CISPA Helmholtz Center for Information Security, Saarbrückem, Germany

Abstract

Constrained Forest Problems (CFPs) as introduced by Goemans and Williamson in 1995 capture a wide range of network design problems with edge subsets as solutions, such as Minimum Spanning Tree, Steiner Forest, and Point-to-Point Connection. While individual CFPs have been studied extensively in individual computational models, a unified approach to solving general CFPs in multiple computational models has been lacking. Against this background, we present the *shell-decomposition algorithm*, a *model-agnostic meta-algorithm* that efficiently computes a $(2+\varepsilon)$-approximation to CFPs for a broad class of forest functions. The shell-decomposition algorithm isolates the problem-specific hardness of individual CFPs in a single computational subroutine, breaking the remainder of the computation into fundamental tasks that are studied extensively in a wide range of computational models. In contrast to prior work, our framework is compatible with the use of *approximate* distances.

To demonstrate the power and flexibility of this result, we instantiate our algorithm for three fundamental, NP-hard CFPs (Steiner Forest, Point-to-Point Connection, and Facility Placement and Connection) in three different computational models (Congest, PRAM, and Multi-Pass Streaming). For constant ε, we obtain the following $(2+\varepsilon)$-approximations in the Congest model:

(1) For Steiner Forest specified via input components (SF–IC), where each node knows the identifier of one of k disjoint subsets of V (the input components), we achieve a deterministic $(2+\varepsilon)$-approximation in $\widetilde{\mathcal{O}}(\sqrt{n}+D+k)$ rounds, where D is the hop diameter of the graph, significantly improving over the state of the art.

(2) For Steiner Forest specified via symmetric connection requests (SF–SCR), where connection requests are issued to pairs of nodes $u, v \in V$, we leverage randomized equality testing to reduce the running time to $\widetilde{\mathcal{O}}(\sqrt{n}+D)$, succeeding with high probability.

(3) For Point-to-Point Connection, we provide a $(2+\varepsilon)$-approximation in $\widetilde{\mathcal{O}}(\sqrt{n}+D)$ rounds.

(4) For Facility Placement and Connection, a relative of non-metric Uncapacitated Facility Location, we obtain a $(2+\varepsilon)$-approximation in $\widetilde{\mathcal{O}}(\sqrt{n}+D)$ rounds.

We further show how to replace the $\sqrt{n}+D$ term by the complexity of solving Partwise Aggregation, achieving (near-)universal optimality in any setting in which a solution to Partwise Aggregation in near-shortcut-quality time is known. Notably, all of our concrete results can be derived with relative ease once our model-agnostic meta-algorithm has been specified. This demonstrates the power of our modularization approach to algorithm design.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Graph algorithms; Theory of computation → Models of computation

Keywords and phrases Distributed Graph Algorithms, Model-Agnostic Algorithms, Steiner Forest

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.25

Related Version *Full Version*: `https://arxiv.org/abs/2407.14536` [10]

Funding *Corinna Coupette*: Supported by *Digital Futures* at KTH Royal Institute of Technology.



© Corinna Coupette, Alipasha Montaseri, and Christoph Lenzen;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The classic approach to determining the computational complexity of a task consists in deriving upper and lower bounds that are as tight as possible in a particular model of computation. On the upper-bound side, this can result in solutions that are specifically engineered toward the chosen computational model, such that the task at hand needs to be reexamined for every relevant model. Worse still, one might end up with fragile solutions that overcome only the challenges specifically represented by the model under study. At first glance, lower bounds might appear more robust in this regard, since they highlight an obstacle that *any* algorithm needs to overcome. However, many such lower bounds are shown for very specific network topologies that are rarely observed in practice. This might lead to a false sense of success in having classified the complexity of a given task – although the lower bound merely indicates that the parameters used to capture the computational complexity of the task are insufficient. In brief, traditional upper and lower bounds bear two limitations: *model specificity* and *existential optimality*.

A textbook illustration of these limitations is provided by the Steiner Forest problem (SF), which generalizes the well-known Steiner Tree problem (ST). In the classic SF formulation using input components (SF–IC), we are given a weighted graph, along with disjoint subsets of nodes $V_1, \dots, V_k \subseteq V$, and the goal is to determine a minimum-weight subgraph spanning each V_i, $i \in [k]$. This general and fundamental connectivity problem has been studied in depth in the classic centralized model of computation [1, 8, 16, 19, 28]. It is of significant practical importance, both due to direct application, see e.g., [36, Sec. 18.5.5], and being the key building block in variants of Steiner-type problems with broad real-world utility, e.g., Steiner Tree reoptimization [5] and Multicommodity Rent-or-Buy [20]. The state of the art in the CONGEST model is due to Lenzen and Patt-Shamir [30], who presented modified and adapted variants of a well-known centralized algorithm by Agrawal et al. [1]. One might suspect that algorithmic techniques underlying these tailored solutions could be transferred to other computational models, but this cannot be readily determined from their specialized solution (*model specificity*). Furthermore, for any fixed topology, there are inputs such that even their fastest algorithm runs for $\widetilde{\Omega}(\sqrt{n})$ rounds.[1] This is known to be necessary in the worst case (*existential optimality*): the Minimum Spanning Tree (MST) problem is a special case of the Steiner Forest Problem with $k = 1$ and $V_1 = V$, to which the $\widetilde{\Omega}(\sqrt{n})$ lower bound by Das Sarma et al. [11] applies. However, the topology of the lower-bound graph is highly contrived, and the technique of using *low-congestion shortcuts* has been demonstrated to overcome the $\sqrt{n}$ barrier for MST in many more realistic settings [13, 14, 24]. These findings motivate us to revisit the Steiner Forest problem, along with a much broader class of connectivity problems, in an attempt to overcome the abovementioned limitations.

Beyond Model Specificity and Existential Optimality

To go beyond of *model specificity* and *existential optimality*, two paradigms have emerged.

First, numerous works have pushed toward what we term *model agnosticism*, developing algorithms that can be readily instantiated in many computational models [6, 7, 21, 32, 35]. The individual steps of these algorithms are either core tasks like distance computation, identifying connected components, and sorting, which are well-studied across a wide range of models, or they are easily implemented at low cost in any notable model. Hence, we call

[1] We use $\widetilde{\mathcal{O}}$ and $\widetilde{\Omega}$ to suppress factors of $\log^{\mathcal{O}(1)} n$.

these algorithms *model-agnostic (meta-)algorithms*. Naturally, model-agnostic algorithms are more robust against model variations, and by allowing us to plug in optimized model-specific subroutines for the core computational problems, they sacrifice little performance over model-specific solutions. Thus, model-agnostic algorithms can be viewed as model-agnostic reductions to more basic computational tasks. Moreover, since these basic tasks can be solved by model-specific subroutines, they improve whenever progress is made on the current performance bottleneck in a given computational model.

Second, *universal optimality*, which was coined in the context of distributed computing, pushes for the design of topology-adaptive algorithms that are asymptotically worst-case optimal on *every* network topology, i.e., when varying input parameters *other* than the underlying communication graph [15, 27, 37].[2] One might argue that this idea is no different than taking into account more parameters, such as the network diameter, the node connectivity, or any other quantity meaningful for the computational task. However, parametrizing complexity by the input graph is extremely general, subsuming a large number of parameters that one might consider, and capturing any *topology-specific obstacle* for the task at hand in a given computational model.

Our Contributions in Brief

In this work, we study a general class of connectivity problems called *Constrained Forest Problems* (CFPs), introduced by Goemans and Williamson [16], through the lenses of *model agnosticism* and *universal optimality*. Intuitively, a CFP is specified by a Boolean function f that indicates, for each node subset $S \subseteq V$, whether it needs to be connected to its complement, i.e., if the output must contain an edge from S to $V \setminus S$. We require f to be *proper*, i.e., (1) $f(V) = 0$, (2) $f(S) = f(V \setminus S)$, and (3) $f(A) = f(B) = 0$ for disjoint $A, B \subseteq V$ implies that $f(A \cup B) = 0$. For example, the Steiner Forest problem can be specified by setting $f(S) = 1$ if and only if, for given disjoint node subsets $V_i \subseteq V$, there exists some $i \in [k]$ such that both $V_i \cap S$ and $V_i \cap (V \setminus S)$ are non-empty.

Model-Agnostic Algorithm for CFPs

We devise the *shell-decomposition algorithm*, a generic approximation algorithm for CFPs that is efficient if f can be evaluated efficiently.

▶ **Theorem 1** (Model-Agnostic Complexity of Constrained Forest Problems). *Given $0 < \varepsilon \leq 1$ and a graph $G = (V, E)$ with polynomially bounded edge weights $c\colon E \to \mathbb{N}$, a $(2+\varepsilon)$-approximation to a CFP with proper forest function $f\colon 2^V \to \{0,1\}$ can (up to book-keeping operations) be obtained at complexity $\widetilde{\mathcal{O}}\left((aSSSP + MST + RPS + FFE)\varepsilon^{-1}\right)$, with the terms in the sum denoting the complexities of solving (1) aSSSP: $(1+\varepsilon)$-approximate Set-Source Shortest-Path Forest, (2) MST: Minimum Spanning Tree, (3) RPS: Root-Path Selection, and (4) FFE: Forest-Function Evaluation for f.*

By *book-keeping operations*, we denote simple steps that extract the input for the next subroutine from the output of the previous ones and can be easily implemented at low complexity, where the complexity measure(s) depend on the model at hand (e.g., round

[2] Note that *instance optimality*, which requires that an algorithm is $\mathcal{O}(1)$-competitive with any other always-correct algorithm on each instance, including the best algorithm for the specific instance, is often unachievable [27]. In particular, one can test whether the input matches an abritrary fixed instance in D rounds. If so, a matching solution can be output without further computation; otherwise, a fallback algorithm is executed.

Table 1 CONGEST results for SF derived from our algorithm, for $\varepsilon \in \Theta(1)$. Deterministic and randomized results are marked with $\mathfrak{D}$ and $\mathfrak{R}$, respectively. Here, n is the number of nodes, D is the (unweighted) hop diameter, s is the shortest-path diameter (which may be different from the weighted diameter), t is the number of terminals, k is the number of SF input components, and Q is the shortcut quality of the input graph, cf. Table 3. In our results, the $\sqrt{n}+D$ terms can be replaced by $T^{PA}n^{o(1)}$, where $T^{PA} \in \mathcal{O}(\sqrt{n}+D)$, $T^{PA} \geq Q$, is the running time of Partwise Aggregation [25]. The $n^{o(1)}$ factors can be removed conditionally on the existence of certain cycle-cover algorithms [35]. For any values of the other parameters, s and t can be linear in n.[4]

Problem	LB		Previous Work			Our Work	
		Ref.	APX	Complexity	Ref.	APX	Complexity
SF(–IC)	$\widetilde{\Omega}(Q)$	[27]	$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(sk+\sqrt{\min\{st,n\}})$	[30]	$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D+k)$
			$\mathcal{O}(\log n)$ $\mathfrak{R}$	$\widetilde{\mathcal{O}}(\min\{s,\sqrt{n}\}+D+k)$			
PPC	$\widetilde{\Omega}(Q)$	[27]		—		$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D)$
FPC	$\widetilde{\Omega}(Q)$	[27]		—		$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D)$

complexity in CONGEST). MST is the special case of SF in which all nodes are to be connected by a lightest possible tree. The task of aSSSP requires us, given $\varepsilon > 0$ and $S \subseteq V$, to compute a forest that preserves distances to S up to a factor of $1+\varepsilon$, which ie equivalent to computing a $(1+\varepsilon)$-approximate shortest-path tree in the graph obtained by identifying all nodes in S. Finally, RPS (a special case of the more general *transshipment* problem) demands that, given a rooted forest $F \subseteq E$ and a set of marked nodes, we select all edges on paths from marked nodes to the roots of their respective trees.

MST, aSSSP, and RPS are well-understood in many models of computation. In contrast, f can be (ab)used to force the evaluation of an arbitrarily hard function g on the entire topology.[3] Thus, Theorem 1 can be viewed as confining the hardness of the task arising from the choice of f to $\mathcal{O}(\varepsilon^{-1}\log n)$ iterations of evaluating $f(C)$ for all $C \in \mathcal{C}$, where $\mathcal{C}$ is a set of disjoint connected components. Because for any such $\mathcal{C}$, one can enforce this evaluation by assigning weight 1 to edges inside each C and a large weight, say n^2, to all other edges, this is the best possible up to an $\mathcal{O}(\varepsilon^{-1}\log n)$ factor.

To illustrate the power and flexibility of our result, we apply our machinery in three models of computation – CONGEST, Parallel Random-Access Machine (PRAM), and Multi-Pass Streaming (MPS) – to three NP-hard CFPs: (1) *Steiner Forest (SF)*; (2) *Point-to-Point Connection (PPC)*, i.e., given $X, Y \subset V$ of equal cardinality, finding a lightest set of edges such that each induced component the number of nodes from X is the same as the number of nodes from Y; and (3) *Facility Placement and Connection (FPC)*, i.e., minimizing the cost of opening facilities at some nodes and connecting a set of clients $C \subseteq V$ to them. Table 1 summarizes our results in CONGEST; for all problems, our PRAM algorithms require $\widetilde{\mathcal{O}}(\varepsilon^{-3}m)$ work and $\widetilde{\mathcal{O}}(\varepsilon^{-3})$ depth, and our MPS algorithms need $\widetilde{\mathcal{O}}(\varepsilon^{-3})$ passes and $\widetilde{\mathcal{O}}(n)$ memory.

[3] For an input graph $G=(V,E)$ with uniform edge weights and any function g mapping such a graph to a desired range, choosing an arbitrary node $v \in V$ and setting $f(S)=1$ if and only if (i) $S=\{v\}$ or $S=V\setminus\{v\}$, and (ii) $g(G)=1$ results in a proper forest function f.

[4] Strictly speaking, the upper bounds from Lenzen and Patt-Shamir [30] are incomparable to ours, as it is possible to construct instances in which s, t, and k are small, yet T^{PA} is not. This corresponds to "easy" instances for the subroutines we use, so one could provide refined bounds. However, this is orthogonal to our goals in this work, as we seek to derive bounds that depend on the topology (V,E) only.

Table 2 CONGEST results for the four input variants of SF, with the notational conventions from Table 1. Regardless of the input representation, on any graph, a lower bound of $\widetilde{\Omega}(Q)$ applies to randomized algorithms; the listed lower bounds arising from the input representation are existential (ex.). As before, $\sqrt{n}+D$ terms can be replaced by $T^{PA}n^{o(1)}$.

Problem	Input	ex. LB	APX	Complexity
SF–IC	Each node v is given its component identifier $\lambda_v \in [k] \cup \{\perp\}$	$\widetilde{\Omega}(k)$ $\mathfrak{R}$	$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D+k)$
SF–CIC	As in SF–IC, but node v knows λ_v *and* $\lvert\{u \in V \mid \lambda_u = \lambda_v\}\rvert$ (if $\lambda_v \neq \perp$)	$\widetilde{\Omega}(k)$ $\mathfrak{D}$	$(2+\varepsilon)$ $\mathfrak{R}$	$\widetilde{\mathcal{O}}(n^{2/3}+D)$
SF–CR	Each node v is given $\mathcal{R}_v \subseteq V \setminus \{v\}$	$\widetilde{\Omega}(t)$ $\mathfrak{R}$	$(2+\varepsilon)$ $\mathfrak{D}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D+t)$
SF–SCR	$\mathcal{R} \subseteq \binom{V}{2}$; each node v knows $\mathcal{R}_v = \{u \in V \mid \{u,v\} \in \mathcal{R}\}$	$\widetilde{\Omega}(t)$ $\mathfrak{D}$	$(2+\varepsilon)$ $\mathfrak{R}$	$\widetilde{\mathcal{O}}(\sqrt{n}+D)$

Approaching Universal Optimality

In our upper bounds for CONGEST, $\sqrt{n}+D$ can largely be replaced by $T^{PA}n^{o(1)}$, where T^{PA} is the running time of an algorithm performing *Partwise Aggregation* (PA) [14, 25] – i.e., given a partition of V into connected subsets of nodes, computing the result of an aggregation function separately on each subset and outputting the result at each of its constituent nodes. Due to the respective hardness results [27], this implies that the running times of our solutions to PPC and FPC are universally optimal up to a factor of $n^{o(1)}$.

For SF–IC, this is true up to the additive term of k. Here, PA is insufficient: Evaluating f requires us to determine, for each set V_i, if it is contained in a single connected component induced by the set of edges that have been selected into the current (partial) solution, but the V_i may not induce connected components in G. Existential lower bounds demonstrate that this obstacle is unavoidable in general [30]. We suspect that studying Partwise Aggregation with disjoint components that may be internally disconnected – i.e., *Disjoint Aggregation* (DA) – is key to characterizing the universal complexity of SF–IC, but leave this to future work, noting that an upper bound of $\mathcal{O}(k+D)$ arises from standard pipelining techniques.

An orthogonal consideration, however, yields surprising results, as summarized in Table 2. For the SF problem, the input representation drives the problem complexity. It is known that if nodes are given the identifiers of other nodes they must connect to in the output to encode connectivity requirements (SF–CR), an existential lower bound of $\widetilde{\Omega}(t)$ applies [30], where t is the number of *terminals*, i.e., nodes that need to be connected to some other node. In our corresponding upper bound, the additive k then becomes an additive t, again resulting in existential but not universal optimality. Interestingly, this picture is turned upside down when inputs are *symmetric* in the following sense: If $u \in V$ knows that it must connect to $v \in V$ by the input, then also v knows that it must connect to u (SF–SCR). In this setting, we can exploit symmetry to efficiently evaluate f using randomized equality testing. This leads to an algorithm running in $\varepsilon^{-3}T^{PA}n^{o(1)}$ rounds, which is close to universal optimality, provided that an efficient partwise-aggregation routine is available. Similarly, we observe that the $\widetilde{\Omega}(k)$ bound can be circumvented if nodes receive not only the identifier of their input component V_i, but also the size $|V_i|$ of this input component (SF–CIC). We then obtain a randomized algorithm running in $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n}+D)+\varepsilon^{-1}n^{2/3})$ time.

We note that a simple adaptation of the lower-bound construction from Lenzen and Patt-Shamir [30] shows the same hardness (i.e., $\widetilde{\Omega}(t)$ for SF–SCR and $\widetilde{\Omega}(k)$ for SF–CIC, respectively) for *deterministic* algorithms, based on a communication-complexity reduction

Table 3 Main notation used in this work. All notation involving nodes and edges makes implicit reference to the graph G.

Symbol	Definition	Meaning
$\lvert S\rvert$		Cardinality of a set S
2^S	$=\{X \mid X \subseteq S\}$	Set of all subsets (i.e., *power set*) of S
$\binom{S}{k}$	$=\{X \subseteq S \mid \lvert X\rvert = k\}$	Set of k-element subsets of S
$[k]$	$=\{i \mid i \in \mathbb{N}, i \leq k\}$	Set of nonnegative integers no larger than k
$[k]_0$	$=\{i \mid i \in \mathbb{N}_0, i \leq k\}$	Set of positive integers no larger than k
G	$=(V, E)$	Graph with node set V and edge set E
n	$=\lvert V\rvert$	Number of nodes
m	$=\lvert E\rvert$	Number of edges
$c(e)$	$\in \mathbb{N} = \{1, 2, \ldots\}$	Cost of edge e
$c(Z)$	$=\sum_{e\in Z} c(e)$	Cost of edge subset $Z \subseteq E$
$p(u, v)$	$=(e_1, \ldots, e_\ell)$ s.t. $\exists(v_1, \ldots, v_{\ell+1})$: $e_i = \{v_i, v_{i+1}\} \in E\ \forall i \in [\ell]$, $u = v_1$, $v = v_{\ell+1}$	ℓ-hop path between u and v (distinct nodes and distinct edges)
$h(u, v)$	$=\min\{i \mid \exists\ p(u, v)$ with $\lvert p(u, v)\rvert = i\}$	Hop (= unweighted) distance between u and v
D	$=\max\{h(u, v) \mid \{u, v\} \in \binom{V}{2}\}$	Hop diameter of G
$d(u, v)$	$=\min\{i \mid \exists\ p(u, v)$ with $c(p(u, v)\}$	Weighted distance between u and v
$P(u, v)$	$=p(u, v)$ with $c(p(u, v)) = d(u, v)$	Shortest path between u and v
s	$=\max\{\min\{\lvert P(u, v)\rvert \mid \{u, v\} \in \binom{V}{2}\}$	Shortest-path diameter of G
$\delta(S)$	$=\{e \in E \mid \lvert e \cap S\rvert = 1\}$	Set of edges with exactly one endpoint in $S \subset V$
$\mathcal{T}$	$=\{v \in V \mid f(\{v\}) = 1\}$	Terminals
t	$=\lvert\mathcal{T}\rvert$	Number of terminals
k		Number of input components in SF–IC

from 2-player equality testing. To the best of our knowledge, this is the first natural example of a provably large gap between the randomized and deterministic complexity in the CONGEST model for a *global* problem.

Structure

After introducing our main definitions in Section 2, we provide a technical overview of our results in Section 3. To conclude the main text, we discuss open questions in Section 4. A full specification of our model-agnostic algorithm and a detailed proof of its correctness are given in Section A. Model descriptions, model-specific results, and further related work are deferred to the additional appendices available in the extended version of this work [10].

2 Preliminaries

We begin by defining our basic notation as well as the class of problems we are interested in.

Basic Notation

Our basic notation is summarized in Table 3.

For a set S, we denote its cardinality by $\lvert S\rvert$, its power set by $2^S = \{X \mid X \subseteq S\}$, and the set of its k-element subsets by $\binom{S}{k}$. We extend functions $f\colon S \to \mathbb{R}$ to subsets $X \subseteq S$ in the natural way by setting $f(X) = \sum_{x\in X} f(x)$, and we write the sets of positive and nonnegative integers no greater than k as $[k] = \{i \in \mathbb{N} \mid i \leq k\}$ resp. $[k]_0 = \{i \in \mathbb{N}_0 \mid i \leq k\}$.

We consider weighted graphs $G = (V, E)$ with $n = |V|$ nodes, $m = |E|$ edges, and edge weights (*edge costs*) $c\colon E \to \mathbb{N}_0$ polynomially bounded in n.[5] Each node is equipped with a unique identifier of $\mathcal{O}(\log n)$ bits, which is also used to break ties. An ℓ-hop *path* from $u \in V$ to $v \in V$, denoted $p(u, v)$, is a sequence of distinct edges $(e_1, \dots e_\ell)$ arising from a sequence of distinct nodes $(v_1, \dots, v_{\ell+1})$ such that $e_i = \{v_i, v_{i+1}\} \in E$ for $i \in [\ell]$, $v_1 = u$, and $v_{\ell+1} = v$. The (unweighted) *hop distance* between u and v is the smallest number of hops needed to go from u to v, i.e., $h(u, v) = \min\{i \mid \exists\, p(u, v) \text{ with } |p(u, v)| = i\}$, and the *hop diameter* of G is $D = \max\{h(u, v) \mid \{u, v\} \in \binom{V}{2}\}$. The (weighted) *shortest-path distance* between u and v is $d(u, v) = \min\{i \mid \exists\, p(u, v) \text{ with } c(p(u, v)) = i\}$. A *shortest path* between u and v, denoted $P(u, v)$, is a path from u to v of length $d(u, v)$. The *shortest-path diameter* s of G is the maximum over all $\{u, v\} \in \binom{V}{2}$ of the minimum number of hops contained in a shortest path from u to v, i.e., $s = \max\{\min\{|P(u, v)| \mid \{u, v\} \in \binom{V}{2}\}\}$. Given a *cut* $(S, V \setminus S)$, the set of edges with exactly one endpoint in S is denoted as $\delta(S) = \{e \in E \mid |e \cap S| = 1\}$.

An event occurring *with high probability* (w.h.p.) has probability at least $1 - 1/n^c$ for a freely chosen constant $c \geq 1$.

Constrained Forest Problems

We are interested in Constrained Forest Problems (CFPs) as introduced by Goemans and Williamson [16].[6] Given a graph $G = (V, E)$ with edge costs $c : E \to \mathbb{N}$ and a function $f : 2^V \to \{0, 1\}$, a CFP asks us to solve the integer program stated as Problem 1, whose dual relaxation is provided as Problem 2.

▶ **Problem 1** (CFP Primal IP).

$$\begin{aligned} &\min \sum_{e \in E} c(e) x_e \\ s.t.\quad &x(\delta(S)) \geq f(S) \quad \forall\, \emptyset \neq S \subset V \\ &x_e \in \{0, 1\} \quad \forall\, e \in E \end{aligned}$$

▶ **Problem 2** (CFP Dual LP).

$$\begin{aligned} &\max \sum_{S \subset V} f(S) y_S \\ s.t.\quad &\sum_{S : e \in \delta(S)} y_S \leq c(e) \quad \forall\, e \in E \\ &y_S \geq 0 \end{aligned}$$

That is, a CFP is a minimization problem whose optimal solution is a *forest*[7] of edges from the input graph G that meets the *constraints* imposed by the *forest function* f. Like Goemans and Williamson [16], we consider CFPs with *proper* functions f, which satisfy (1) *zero*, i.e., $f(V) = 0$, (2) *symmetry*, i.e., $f(S) = f(V \setminus S)$, and (3) *disjointness* (also called *maximality* [17]), i.e., if $A \cap B = \emptyset$ for two sets A and B, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.[8]

For a CFP with forest function f, a node v is called a *terminal* if $f(\{v\}) = 1$. Given a forest function f, we denote the set of terminals as $\mathcal{T} = \{\{v\} \mid v \in V, f(\{v\}) = 1\}$ and the number of terminals as $t = |\mathcal{T}|$.

[5] Assuming polynomially bounded edge weights allows us to encode polynomial sums of edge weights with $\mathcal{O}(\log n)$ bits, which means that we can encode edge weights in a single message (CONGEST) or memory word (PRAM and MPS). Zero-weight edges arise naturally when simulating contractions in the distributed setting. We can handle them by scaling all non-zero edge weights by n/ε (where w.l.o.g., $1/\varepsilon$ is polynomially bounded as well), and assigning weight 1 to all previously zero-weight edges.

[6] To simplify the technical exposition, like Goemans and Williamson [16], we disallow zero-weight edges. However, it is straightforward to extend our approach to zero-weight edges by scaling edge weights as discussed in Footnote 5.

[7] For any cycle and node set S, $\delta(S)$ cannot contain exactly one edge from the cycle. Hence, from any solution with a cycle, we can remove an edge.

[8] We also assume that f is *nontrivial*, i.e., that there exists at least one $S \subset V$ such that $f(S) = 1$.

Specific Constrained Forest Problems

To demonstrate the flexibility of our approach, we consider Survivable Network Design Problems (SNDPs) that originate from three different real-world challenges.

▶ **Definition 2** (Steiner Forest (SF)). *Given a partition of the terminals $\mathcal{T} = V_1 \dot\cup \ldots \dot\cup V_k$, find a minimum-cost edge subset connecting the nodes in each input component. The corresponding forest function evaluates to 1 on $S \subseteq V$ if and only if there is $i \in [k]$ such that $\emptyset \neq V_i \cap S \neq V_i$. We distinguish between several input representations:*

SF–CR *Terminal v is given the identifiers of other terminals as* connection requests *$R_v \subseteq \mathcal{T}$.*

SF–SCR *As SF-CR, but connection requests are symmetric, i.e., $u \in R_v \Leftrightarrow v \in R_u$.*

SF–IC *Terminal $v \in V_i$ is given a unique identifier (of size $\mathcal{O}(\log n)$) for its* input component*.*

SF–CIC *As SF-IC, but $v \in V_i$ is also given the* cardinality *$|V_i|$ of its input component as input.*

SF has practical relevance especially in infrastructure development [1], with MST ($\mathcal{T} = V_1 = V$) and ST ($\mathcal{T} = V_1 \subseteq V$) as special cases. It is NP-complete [29] and APX-hard [9].

▶ **Definition 3** (Point-to-Point Connection (PPC)). *Given a set of sources $X \subset V$ and a set of targets $Y \subset V$, find a minimum-cost edge subset such that in each connected component, the number of sources equals the number of targets. That is, for $S \subseteq V$, $f(S) = 1$ if and only if $|S \cap X| \neq |S \cap Y|$.*

PPC is motivated by challenges from circuit switching and VLSI design, and the problem is NP-complete [31].

Our last problem is Facility Placement and Connection (FPC), an NP-complete facility-location-type problem arising, e.g., in operations research. Intuitively, FPC can be stated as follows.

▶ **Definition 4** (FPC [intuitive]). *Given for each node $v \in V$ an opening cost $o_v \in \mathbb{N}$ and indication whether it is in the set of clients $C \subseteq V$, identify a subset $O \subseteq V$ of facilities to open and an edge set $F \subseteq E$ such that each client is connected to a facility by F, minimizing $\sum_{v \in O} o_v + \sum_{e \in F} c(e)$.*

To turn this task into a CFP matching our framework, we add one additional node $s \notin V$ and, for each $v \in V$, an edge $\{v, s\}$ of weight $c(\{v, s\}) = o_v$. The task then becomes to determine a (low-weight) Steiner Tree spanning $C \cup \{s\}$, i.e., the special case of SF with $k = 1$.

▶ **Definition 5** (FPC [rephrased]). *Given, for each node $v \in V$, an opening cost $o_v \in \mathbb{N}$ and an indication whether it is in the set of clients $C \subseteq V$, solve ST on $G = (V \dot\cup \{s\}, E \dot\cup \{\{v, s\} \mid v \in V\})$, with edge costs of $c(e)$ for $e \in E$ and $c(\{v, s\}) = o_v$ for $v \in V$, where the terminals are $\mathcal{T} = C \cup \{s\}$.*

Even in Congest, we can solve ordinary ST instances efficiently, regardless of the specific input representation. However, the *virtual node* s and its incident edges need to be *simulated* in the chosen model of computation. This is trivial in PRAM (simply modify the input representation in parallel) and Multi-Pass Streaming (since we use $\Omega(n \log n)$ bits of memory anyway) but nontrivial in Congest. For Congest, we show that we can simulate the virtual node efficiently using Partwise Aggregation.

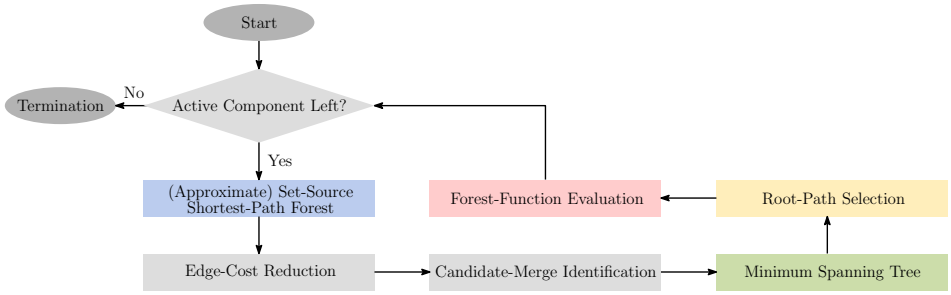


Figure 1 Overview of our model-agnostic shell-decomposition algorithm for approximating Constrained Forest Problems. Main tasks are colored; book-keeping operations are shaded in gray.

Partwise Aggregation and Shortcut Quality

Finally, we introduce a subroutine that we use as a black box to achieve near-universal optimality in cases where efficient solutions are known.

▶ **Definition 6** (Partwise Aggregation [13]). *For disjoint node sets $V_1, \dots, V_k \subseteq V$, suppose that V_i induces a connected subgraph. In the* Partwise Aggregation (PA) *problem, each $v \in V_i$ is given a unique identifier for V_i (of size $\mathcal{O}(\log n)$) and a second $\mathcal{O}(\log n)$-bit value $x(v) \in X$. For a specified associative and commutative operator $\bigoplus\colon X \times X \to X$, for each $i \in [k]$ and each $v \in V_i$, v needs to compute its output $\bigoplus_{w \in V_i} x(w)$.*

▶ **Definition 7** (Shortcut Quality). *The* shortcut quality *of G, denoted by Q, is the maximum over all feasible operators $\bigoplus$ and partitions of V of the minimum number of rounds in which a* Congest *algorithm with knowledge of the full topology can solve Partwise Aggregation. Put differently, the algorithm may preprocess the graph and the operator, but must then compute the output within Q rounds after the nodes have been given their inputs to the PA instance.*

Haeupler et al. [27] show that $\widetilde{\Omega}(Q)$ is a lower bound for MST (i.e., SF and ST with $k = 1$ and $V_1 = V$) and shortest s-t path (the special case of PPC with $|X| = |Y| = 1$) – regardless of the approximation ratio and also for randomized Las Vegas algorithms, i.e., those that guarantee a feasible output. They further prove that PA can be solved in $\widetilde{\mathcal{O}}(Q)$ rounds in the *Supported* Congest model, which is Congest with the unweighted graph topology given as part of the input.

3 Technical Overview

In this section, we outline our techniques and discuss the technical challenges to be overcome. We also use this opportunity to discuss the most relevant related work in context.

Model-Agnostic Algorithm for Proper Constrained Forest Problems

Our shell-decomposition algorithm, depicted in Figure 1, is based on the primal-dual formulation for general CFPs given by Goemans and Williamson [16] (GW algorithm), restated as Algorithm 1, which yields a $(2 - 2/t)$-approximation [16] for CFPs with proper forest functions. Our algorithm can also be seen as a model-agnostic generalization of the algorithm by Agrawal et al. [1], ported to the distributed setting by Lenzen and Patt-Shamir [30]. These algorithms have also been called *moat-growing* algorithms [3, 18, 30].

Intuitively, our algorithm operates as follows. We maintain connected *components* $\mathcal{C}$, initialized to the singletons $\mathcal{C} = \{\{v\} \mid v \in V\}$. A component $C \in \mathcal{C}$ is *active* if $f(C) = 1$. The algorithm concurrently grows balls around all *active components*, with respect to the

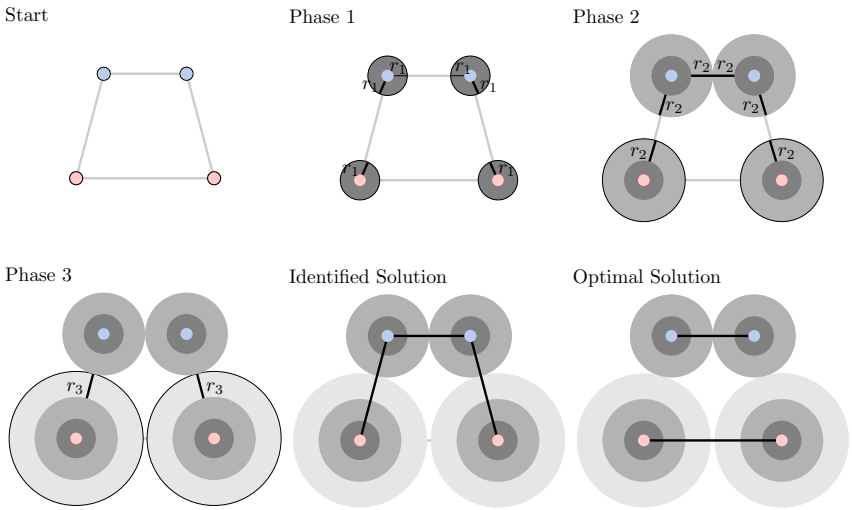


Figure 2 Illustration of our *shell-decomposition argument* on a small instance of Steiner Forest (SF–IC). Gray lines indicate original edges, black lines indicate (parts of) selected edges, black circle linings indicate active components, and node colors indicate input components. The illustrations provided by Goemans and Williamson [16] (Figs. 2–5) are stylistically similar, but our drawings clarify the phase-wise charging argument underlying our shell-decomposition algorithm.

metric induced by the then-current edge costs c'. In the growth process, two balls can touch only when at least one of their associated components is active, and only when the balls of *two active components* C, C' touch, adding edges to merge the components can make the resulting component inactive – otherwise, i.e., assuming $f(C \cup C') = f(C') = 0$, symmetry implies $f(V \setminus (C \cup C')) = 0$, disjointness implies $f(V \setminus C) = f((V \setminus (C \cup C')) \cup C') = 0$, and using symmetry again we get $f(C) = 0$, a contradiction.

As illustrated in Figure 2, until the balls around two components touch, they are disjoint, witnessing that the dual problem has a solution with weight larger than the product of the current radius times the number of currently active components. Accordingly, when merging active components, we can afford to connect the terminals by adding a shortest path between them to the primal solution, paying a cost of at most twice the radius at merge. Because each merge we perform reduces the number of active components by at least one, the ball growth always witnesses sufficient additional weight in a dual solution to pay for future merges up to an approximation factor of 2. Upon termination, i.e., when all components are inactive, the set of edges we selected constitutes a feasible solution.

The intuition sketched above already suggests that the ball-growing process allows for substantial concurrency. To achieve high efficiency across the board in various models, our shell-decomposition algorithm performs several modifications to the centralized GW algorithm, which originally assumes a *final filtering step* to eliminate unneeded edges from the solution, as well as *exact distances* and *immediate forest-function evaluation*.

(A) *Incremental Solution-Set Construction.* We merge active components that touch *regardless* of whether this ultimately turns out to be necessary to satisfy connectivity requirements. This does not impact the approximation guarantee, which was implicit already in the contribution by Agrawal et al. [1] and is now made explicit in our reformulation of the GW framework [16]. As a result, we need to determine if f imposes further connectivity requirements only as often as we iterate through the loop in Figure 1.

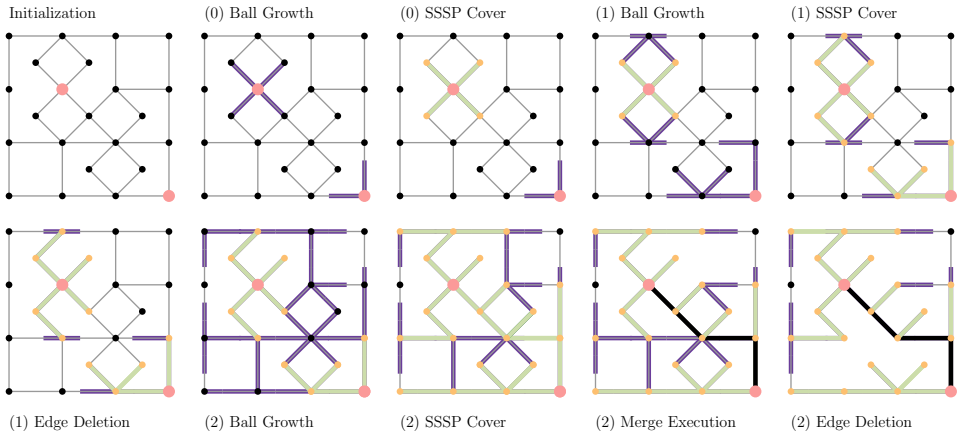


Figure 3 Operation of our shell-decomposition algorithm (Algorithm 2) on an s-t-shortest-path instance seeking to connect the red nodes, starting with $r_0 = 1/2$ (cf. Line 7), and working over phases 0, 1, and 2. Panels are labeled with their phase number and the illustrated step of Algorithm 2. Nodes absorbed by the SSSP forest are drawn in orange, edge-cost reduction is indicated in purple, edges selected into the SSSP forest are marked in green, and edges selected into the solution are marked in black. Distance approximations and deferred forest-function evaluation are not shown.

(B) *Approximate Distance Computations.* At the cost of a factor of $1 + \varepsilon$ in the approximation ratio, we replace exact distance computations with $(1 + \varepsilon)$-approximate distance computations. The challenge here is to ensure that we do not violate the dual constraints when charging dual variables (corresponding to cuts) based on the progress of the primal solution. This can be achieved by constructing the dual solution in the true metric space, rather than reusing the approximate distances leveraged by the primal solution. In contrast to the other modifications, this requires a comparatively involved argument, and it is the main technical novelty and contribution in this part of our work.

(C) *Deferred Forest-Function Evaluation.* Again at the cost of a factor of $1 + \varepsilon$ in the approximation ratio, we let components grow to radii that are integer powers of $1 + \varepsilon$ before reevaluating our forest function to update their activity status. This technique was introduced by Lenzen and Patt-Shamir [30] for the Steiner Forest problem specifically; we show its general correctness in the context of the GW algorithm. Using this technique, we can limit the number of loop iterations in Figure 1 to $\mathcal{O}(\varepsilon^{-1} \log n)$ (assuming polynomially bounded edge weights).

The pseudocode of the resulting model-agnostic shell-decomposition algorithm is given as Algorithm 2. Here, to increase clarity and highlight the primal-dual nature of our algorithm, we also compute and output the lower bound LB on the cost of an optimal solution, which is not required to determine the output forest F. An example execution of Algorithm 2 is depicted in Figure 3. In Section A, we prove that the algorithm maintains an approximation ratio of $2 + \varepsilon$.

Leveraging the modifications specified in Items A–C, to derive concrete algorithms in specific models of computation, what remains is to implement the individual steps in Figure 1. Up to simple book-keeping operations, this entails four main tasks (colored boxes in Figure 1):

(1) *(Approximate) Set-Source Shortest-Path Forest (aSSSP).* This is essentially computing a single-source shortest-path tree with a virtual source node, so we can plug in state-of-the-art algorithms for each model of interest [4, 35].

Algorithm 1 GW Algorithm for $(2 - 2/t)$-Approximation of Constrained Forest Problems [16].

Input: An undirected graph $G = (V, E)$, edge costs $c : E \to \mathbb{N}$, and a proper function f
Output: A forest F and a value LB

$F' \leftarrow \emptyset$
$LB \leftarrow 0$ // Implicitly set $y_S \leftarrow 0$ for all $S \subset V$
$\mathcal{C} \leftarrow \{\{v\} \mid v \in V\}$
foreach $v \in V$ **do**
  $r(v) \leftarrow 0$
while $\exists C \in \mathcal{C} : f(C) = 1$ **do**
  $E_{\mathcal{C}} \leftarrow \{\{i,j\} \in E \mid i \in C_i \in \mathcal{C}, j \in C_j \in \mathcal{C}, C_i \neq C_j, f(C_i) + f(C_j) > 0\}$
  $\eta, e \leftarrow \min_{e=\{i,j\} \in E_{\mathcal{C}}} \operatorname{argmin} \left\{ \frac{c'(e)}{f(C_i)+f(C_j)} \mid c'(e) = c(e) - r(i) - r(j) \right\}$
  $F' \leftarrow F' \cup \{e\}$
  foreach $v \in C_r \in \mathcal{C}$ **do**
    $r(v) \leftarrow r(v) + \eta \cdot f(C_r)$
  $LB \leftarrow LB + \eta \cdot \sum_{C \in \mathcal{C}} f(C)$ // Implicitly set $y_C \leftarrow y_C + \eta \cdot f(C)\ \forall C \in \mathcal{C}$
  $\mathcal{C} \leftarrow (\mathcal{C} \cup \{C_i \cup C_j\}) \setminus \{C_i, C_j\}$
$F \leftarrow \{e \in F' \mid f(N) = 1 \text{ for some connected component } N \text{ of } (V, F' \setminus \{e\})\}$
return F, LB

Algorithm 2 Our Shell-Decomposition Algorithm for $(2+\varepsilon)$-Approximation of Constrained Forest Problems. To obtain the desired approximation guarantee, we choose $\varepsilon', \varepsilon'' \leq \varepsilon/4$.

Input: A graph $G = (V, E)$, edge costs $c : E \to \mathbb{N}$, and a proper forest function f
Output: A forest F and a value LB

$F \leftarrow \emptyset$
$\mathcal{C} \leftarrow \{\{v\} \mid v \in V\}$
$\mathcal{T}^1 \leftarrow \{v \mid f(\{v\}) = 1\}$ // At the beginning, $|\mathcal{T}^1| = t$
foreach $e \in E$ **do**
  $c'(e) \leftarrow c(e)$ // c' keeps track of *reduced* costs
$LB \leftarrow 0$
$r \leftarrow \frac{\varepsilon''}{4}$ // For upper bound (edge selection), we could start with any $r \leq 1/2$
while $\exists C \in \mathcal{C} : f(C) = 1$ **do**
  $U \leftarrow$ Union of $(1+\varepsilon')$-approx. balls of radius r around nodes in $\mathcal{T}^1$ under edge costs c'[9]
  $F' \leftarrow (1+\varepsilon')$-approx. SSSP forest for edge cost c' with set source $\mathcal{T}^1$, restricted to U
  foreach $e \in E$ **do**
    $c'(e) \leftarrow \max\{0, c'(e) - c_U(e)\}$ // $c_U(e) = \sum_{v \in e} \max\{r - d_{F'}(\mathcal{T}^1, v), 0\}$
  $\mathcal{C}_U \leftarrow$ Connected components of (V, F') containing a (unique) $\tau \in \mathcal{T}^1$
  $M \leftarrow \{\{u,v\} \in E \mid u \in C_u \in \mathcal{C}_U, v \in C_v \in \mathcal{C}_U, C_u \neq C_v, c'(\{u,v\}) = 0\}$ // Merge cand.
  $A \leftarrow$ Arbitrary subset of M such that $F' \cup A$ is a forest spanning $(V, F' \cup M)$
  $F \leftarrow F \cup A \cup \bigcup_{v \in e \in A} \{p_v \mid p_v \text{ is the (unique) path in } F' \text{ from } v \text{ to some } \tau \in \mathcal{T}^1\}$
  // Merge
  $E \leftarrow (E \setminus \{e \in E \mid c'(e) = 0\}) \cup F' \cup A$ // Remove unneeded edges contained in U
  $\mathcal{C} \leftarrow$ Connected components of (V, F) // Update connected components
  $\mathcal{T}^1 \leftarrow \{\min\{v \in C \mid f(\{v\}) = 1\} \mid f(C) = 1, C \in \mathcal{C}\}$ // Update active terminals
  $LB \leftarrow LB + (1+\varepsilon')^{-1} r |\mathcal{T}^1|$
  $r \leftarrow (1+\varepsilon'')r$ // Update ball radius
return F, LB

[9] U as used in Line 9 contains all parts of edges in a $(1+\varepsilon)$-approximate SSSP forest that lie at distance at most r from their respective roots. Note that U can also contain *parts of* edges, whereas F' (Line 10), a $(1+\varepsilon)$-approximate SSSP forest truncated to U, only contains full edges.

(2) *Minimum Spanning Tree (MST).* This is another well-studied task for which near-optimal solutions are known in all prominent models [12, 27, 33, 34].

(3) *Root-Path Selection (RPS).* Here, we are given a forest rooted at sources (where each node knows its parent in its tree and its closest source), along with a number of marked nodes. Our goal is to select the edges on the path from each marked node to its closest source. This problem can be straightforwardly addressed by solving a much more general flow problem: *(Approximate) Transshipment*, in which flow demands are to be satisfied at minimum cost. Again, we can plug in state-of-the-art algorithms for each model of interest [4, 35]. Note that in our case, the solution is restricted to containing the edges of a predetermined forest, such that the solution is unique, edge weights play no role, and the challenge becomes to determine the feasible flow quickly.

(4) *Forest-Function Evaluation (FFE).* The remaining subtask is to assess if $f(C) = 1$, for each component $C \subset V$ in a set of disjoint, internally connected components $\mathcal{C}$– i.e., to determine which of the components still need to be connected to others. This is the only step that depends on f, requiring an implementation of f matching the given model of computation.

Note that f is an arbitrary proper function, so evaluating it can be arbitrarily hard. Thus, our algorithm confines the hardness of the task that comes from the choice of f to $O(\varepsilon^{-1} \log n)$ evaluations of $f(C)$ for disjoint connected components $C \in \mathcal{C}$ (cf. Item A). In contrast, the other three subtasks can be solved by state-of-the-art algorithms from the literature in a black-box fashion.

To illustrate the power of our result, we apply our machinery in three models of computation – Congest, Parallel Random-Access Machine (PRAM), and Multi-Pass Streaming (MPS) – to three Constrained Forest Problems. Our results follow from Theorem 1 with (1) the referenced results on aSSSP, MST, and RPS, (2) model- and problem-specific subroutines for FFE, and (3) model-specific subroutines for book-keeping operations.

In Table 1, we highlight the improvements over the state of the art achieved for our example problems by instantiating the shell-decomposition algorithm in the Congest model.

(I) *Steiner Forest (SF).* From $\widetilde{\mathcal{O}}(\varepsilon^{-1}(sk+\sqrt{\min\{st, n\}}))$ time[10] for a $(2+\varepsilon)$-approximation and $\widetilde{\mathcal{O}}(\min\{s, \sqrt{n}\} + D + k)$ time for an $\mathcal{O}(\log n)$-approximation to SF–IC obtained by Lenzen and Patt-Shamir [30] to $(2+\varepsilon)$-approximations in time (1) $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n} + D) + \varepsilon^{-1}k)$ for SF–IC, (2) $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n} + D))$ for SF–SCR, and (3) $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n} + D) + \varepsilon^{-1} \min\{n^{2/3}, k\})$ for SF–CIC. For SF–CR, we incur the same additive running-time overhead of $\mathcal{O}(t)$ as Lenzen and Patt-Shamir [30], matching the existential lower bound they showed.

(II) *Point-to-Point Connection (PPC).* We are not aware of prior work providing Congest algorithms for the PPC problem. Here, we obtain a $(2+\varepsilon)$-approximation in $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n} + D))$ time.

(III) *Facility Placement and Connection (FPC).* We do not know of any existing Congest algorithms for the FPC problem, and we realize a $(2+\varepsilon)$-approximation in $\widetilde{\mathcal{O}}(\varepsilon^{-3}(\sqrt{n} + D))$ time.

We also derive algorithms for SF, PPC, and FPC in the PRAM and Multi-Pass Streaming models, taking $\widetilde{\mathcal{O}}(\varepsilon^{-3}m)$ work and $\widetilde{\mathcal{O}}(\varepsilon^{-3})$ depth in the PRAM model, as well as $\widetilde{\mathcal{O}}(\varepsilon^{-3})$ passes and $\widetilde{\mathcal{O}}(n)$ space in the Multi-Pass Streaming model. To the best of our knowledge,

[10] Recall that n denotes the number of nodes, k the number of input components, t the number of terminals, D the unweighted (hop) diameter, and s the shortest-path diameter. Both t and s can be up to $\Omega(n)$, regardless of k or D.

these algorithms are either the first ones to perform these tasks in their respective models or they cover more general classes of instances than the state of the art (see extended version). Notably, we obtain our diverse results with relative ease once the analysis of our model-agnostic shell-decomposition algorithm is in place – which contrasts with the challenges of directly designing specific algorithms for specific problems in specific models.

Taking Shortcuts

In the CONGEST model, the $\sqrt{n}$ term in the complexity is due to the fact that, in general, it is not possible to solve *Partwise Aggregation* (PA) in $\widetilde{o}(\sqrt{n})$ rounds. As formalized in Definition 6, PA denotes the task of performing an aggregation and broadcast operation on each subset in a partition of V that induces connected components [14]. We can leverage PA, *inter alia*, to determine a leader and distribute its ID, or to find and make known a heaviest outgoing edge, for each component (a.k.a. *part*) in parallel. PA-type operations are key to efficient MST construction, and any T^{PA}-round solution to Partwise Aggregation lets us solve MST in $\widetilde{\mathcal{O}}(T^{PA} + D) = \widetilde{\mathcal{O}}(T^{PA})$ rounds [13, 14].

As MST computation is a CFP, it might not surprise that Partwise Aggregation can serve as a key subroutine for other CFPs in the CONGEST model as well. We show that the $\sqrt{n}$ term in the complexity of CFPs can be replaced by T^{PA}. Since this is already known for $(1+\varepsilon)$-approximate Set-Source Shortest-Path Forest [35], Minimum Spanning Tree [13], and $(1+\varepsilon)$-approximate Transshipment [35] (which can be used to solve Root-Path Selection),[11] again we need to show this for Forest-Function Evaluation only. This is straightforward for PPC and FPC, and simple algorithms evaluate the forest function for SF-IC and SF-CR in $\mathcal{O}(T^{PA} + k)$ and $\mathcal{O}(T^{PA} + t)$ rounds, respectively.

The substantial literature on *low-congestion shortcuts* provides a large array of results on solving Partwise Aggregation in time comparable to the *shortcut quality* of the input graph, i.e., the best possible running time T^{PA} for Partwise Aggregation [2,14,15,22,23,26,27,35,37]. In particular, $T^{PA} \in \widetilde{\mathcal{O}}(D)$ if the input graph does not contain a fixed $\widetilde{\mathcal{O}}(1)$-dense minor, without precomputations or further knowledge of G [14]. Examples of graphs without $\widetilde{\mathcal{O}}(1)$-dense minors are planar graphs and, more generally, graphs of bounded genus. Moreover, in *Supported* CONGEST (where (V, E) is known – or rather, can be preprocessed), $T^{PA} \in \widetilde{\mathcal{O}}(Q)$, i.e., within a polylogarithmic factor of the optimum. Due to the modular structure of our results, any future results on Partwise Aggregation and low-congestion shortcuts will automatically improve the state of the art for CFPs in the CONGEST model.

The Quest for Universal Optimality

In the CONGEST model, it is known that even on a *fixed* network topology, $\widetilde{\Omega}(T^{PA})$ rounds are required to obtain *any* non-trivial approximation for a large class of problems. This class includes MST, a special case of ST, which reduces to FPC by making all but the opening cost of a single node very high, and shortest s-t-path [27], a special case of PPC. Thus, this *universal* lower bound applies to our example tasks as well. In particular, our results on PPC and FPC have almost universally tight running times.

In contrast, the additive terms of k and t in the running times of our algorithms for SF-IC and SF-CR, respectively, are only *existentially* optimal [30]: The lower-bound graph – a double star – has shortcut quality $\mathcal{O}(1)$, but in a fully connected graph, it is trivial to evaluate f for

[11] The results for approximate Set-Source Shortest-Path Forest and approximate Transshipment are conditional on the existence of fast $(\widetilde{\mathcal{O}}(1), \widetilde{\mathcal{O}}(1))$-cycle-cover algorithms; otherwise, we incur an $n^{o(1)}$ overhead (see Table 1).

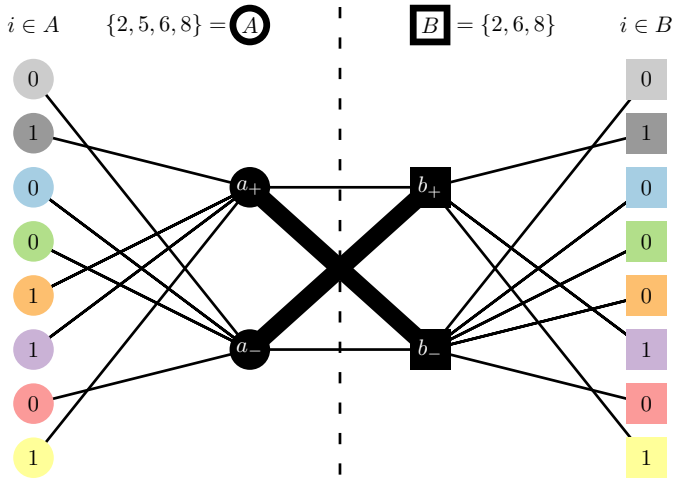


Figure 4 Example of a lower-bound graph used to reduce Equality Testing to Steiner Forest, for an instance with $n = 8$. Node colors indicate input components (SF–CIC) resp. connection requests (SF–SRC), and node shapes indicate which nodes are simulated by which player.

all current components in two rounds. This motivates us to split Forest-Function Evaluation for Steiner Forest into two parts: (1) determining, for each input component $V_i \subseteq V$ or connection request $r \in \mathcal{R}$, respectively, whether the specified connectivity requirements are *met*, and (2) determining, for each current component $C \in \mathcal{C}$, whether it contains a terminal whose connectivity requirements are *not met*. While the second part can be solved via standard Partwise Aggregation, the first part requires aggregating information within k (or t) disjoint components that may be *internally disconnected*. We call this task *Disjoint Aggregation on p parts*, DA(p), achieve the trivial bound $T^{DA(p)} \in \mathcal{O}(D+p)$ via pipelining over a BFS tree, and hypothesize that the universal complexity of DA(p) merits further investigation in future work.

As an orthogonal approach, we explore the effect of the input specification on our SF results. The assumptions that (1) pairs of terminals know that they need to be connected (SF–SCR), or (2) the size of each input component is known to its constituent terminals (SF–CIC) both are plausible, and they change the basis of the applicable existential lower bounds from 2-party *set disjointness* [30] to 2-party *equality*, as depicted in Figure 4. We provide the details in the extended version.

Assuming SF–SCR, we show how to achieve a running time of $\widetilde{\mathcal{O}}(\min\{T^{PA}n^{o(1)}, \sqrt{n}+D\})$ using efficient randomized 2-party equality testing. We sketch a solution assuming *shared* randomness here; standard techniques achieve the same without shared randomness. For each terminal-request pair $\{u,v\}$, we flip a fair independent coin and denote the result by $c_{\{u,v\}} \in \{0,1\}$. Now each component C aggregates $\sum_{u\in C}\sum_{v\in\mathcal{R}_v} c_{\{u,v\}} \bmod 2$. Observe that if $u,v \in C$ for request pair $\{u,v\}$, then for SF–SCR, it holds that $v \in \mathcal{R}_u$ and $u \in \mathcal{R}_v$. Hence, if C contains, for each request pair, either both terminals or none of the terminals, then the sum is guaranteed to be 0 modulo 2. Otherwise, fix a request pair $\{u,v\}$ with $u \in C$ and $v \in V \setminus C$. After evaluating the sum up to coin $c_{u,v}$, it is either 0 or 1, and hence, by independence of $c_{u,v}$, with probability $1/2$, the sum is 1 modulo 2. Therefore, performing the process $\mathcal{O}(\log n)$ times in parallel, we can distinguish between $f(C) = 0$ and $f(C) = 1$ with high probability. This computation can be performed by a single aggregation, where the aggregation operator $\bigoplus$ is given by bit-wise addition modulo 2.

Our strategy for SF–CIC is similar, but results in a much weaker bound of $\widetilde{\mathcal{O}}(n^{2/3}+D)$; our main point here is to demonstrate that the $\widetilde{\Omega}(k)$ can be beaten. We distinguish three cases. (1) Components C of size at most $n^{2/3}$ are spanned by a rooted tree of size $n^{2/3}$. Here, we can aggregate the terminal counts, for all input components, within C at C's root in $\mathcal{O}(n^{2/3})$ rounds to determine activity status. (2) For each input component of size at least $n^{1/3}$, we globally aggregate if there are two distinct component IDs with a terminal from that input component. This requires one aggregation for each such input component, all of which can be completed within $\mathcal{O}(n^{2/3}+D)$ rounds via a BFS tree, as there can be at most $n^{2/3}$ input components of this size. (3) For input components of size $s < n^{1/3}$, each component C of size larger than $n^{2/3}$ uses the same strategy as for SF–SCR. However, only input components of the same size can be handled in a single aggregation, as the summation is now modulo s. Hence, $\widetilde{\mathcal{O}}(n^{1/3})$ aggregations by at most $n^{1/3}$ components of size larger than $n^{2/3}$ are required, which can again be performed in $\widetilde{\mathcal{O}}(n^{2/3}+D)$ rounds.

4 Discussion

In this work, we presented a general model-agnostic framework for the $(2+\varepsilon)$-approximation of Constrained Forest Problems (CFPs) and demonstrated its utility on three NP-hard CFPs in three models of computation. We conclude with a number of open questions – beyond applying our framework to other graph problems and computational models – in increasing order of generality:

(Q1) **Can the running time of SF–CIC in CONGEST be improved to nearly T^{PA}?**

(Q2) Many of our results are near-universally optimal in CONGEST, even for Las Vegas algorithms – i.e., randomized with guaranteed success – but our algorithms for SF–SCR and SF–CIC are Monte Carlo. Due to existential lower bounds based on the communication complexity of 2-party equality, this is required to (always) achieve small running times w.h.p.
Is $\widetilde{\Omega}(Q)$ also a lower bound for Las Vegas CONGEST algorithms?

(Q3) **How hard is Disjoint Aggregation, i.e., how can we characterize $T^{DA(p)}$?**

(Q4) The FPC problem minimizes the sum of opening and forest costs, disregarding the (often distance-based) *connection costs* considered in other facility-location-type problems.
Does our approach to FPC generalize to problems with connection costs?

(Q5) As we reduce most of our tasks to few PA instances, in CONGEST, $T^{PA} \approx Q$ rounds are both necessary and sufficient to achieve near-universal optimality. Since PA can be solved in $\mathcal{O}(n)$ work and $\mathcal{O}(\log n) = \widetilde{\mathcal{O}}(1)$ depth in PRAM, PA-based algorithms also yield good solutions in PRAM. However, in the streaming model, while PA can be solved in $\widetilde{\mathcal{O}}(n)$ memory and two passes, it is unclear if this yields optimal results (unless we insist that the output needs to be held in memory), as we are mostly limited to existential $\widetilde{\Omega}(\sqrt{n})$ lower bounds.
Are there $\tilde{o}(n)$-memory streaming algorithms with $\widetilde{\mathcal{O}}(1)$ passes? Better yet: Is there an analog to universal optimality in the MPS model?
Note that T^{PA} seems inadequate as a parameter here, as each part will require *some* memory for a few-pass implementation, but there may be $\Omega(n)$ parts.

(Q6) **Does our approach generalize to CFPs on hypergraphs?**

(Q7) Beyond proper functions, the primal-dual method has proven useful for *uncrossing* functions [17]. One of the main features of optimization problems with uncrossing functions is that they are guaranteed to feature an optimal dual solution that is *laminar*.
Can our approach be extended to uncrossing or other non-proper functions?

References

1 Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. doi:10.1137/S0097539792236237.

2 Ioannis Anagnostides, Christoph Lenzen, Bernhard Haeupler, Goran Zuzic, and Themis Gouleakis. Almost universally optimal distributed laplacian solvers via low-congestion shortcuts. *Distributed Comput.*, 36(4):475–499, 2023. doi:10.1007/s00446-023-00454-0.

3 Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for prize-collecting steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011. doi:10.1137/090771429.

4 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM J. Comput.*, 50(3):815–856, May 2021. doi:10.1137/19M1286955.

5 Davide Bilò. New algorithms for steiner tree reoptimization. *Algorithmica*, 86(8):2652–2675, 2024. doi:10.1007/S00453-024-01243-2.

6 Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Nearly optimal communication and query complexity of bipartite matching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1174–1185. IEEE, 2022. doi:10.1109/FOCS54457.2022.00113.

7 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of $(\Delta+1)$ coloring in congested clique, massively parallel computation, and centralized local computation. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 471–480. ACM, 2019. doi:10.1145/3293611.3331607.

8 Chandra Chekuri and F. Bruce Shepherd. Approximate integer decompositions for undirected network design problems. *SIAM J. Discret. Math.*, 23(1):163–177, 2008. doi:10.1137/040617339.

9 Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theor. Comput. Sci.*, 406(3):207–214, 2008. doi:10.1016/j.tcs.2008.06.046.

10 Corinna Coupette, Alipasha Montaseri, and Christoph Lenzen. Model-agnostic approximation of constrained forest problems, 2024. doi:10.48550/arXiv.2407.14536.

11 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012. doi:10.1137/11085178X.

12 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

13 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219. SIAM, 2016. doi:10.1137/1.9781611974331.ch16.

14 Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 213–221. ACM, 2021. doi:10.1145/3465084.3467935.

15 Mohsen Ghaffari and Goran Zuzic. Universally-optimal distributed exact min-cut. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 281–291. ACM, 2022. doi:10.1145/3519270.3538429.

16 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

17 Michel X Goemans and David P Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In *Approximation Algorithms for NP-Hard Problems*, pages 144–191. PWS Publishing Co., 1996.

18 Anupam Gupta and Amit Kumar. A constant-factor approximation for stochastic steiner forest. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 659–668. ACM, 2009. doi:10.1145/1536414.1536504.

19 Anupam Gupta and Amit Kumar. Greedy algorithms for steiner forest. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 871–878. ACM, 2015. doi:10.1145/2746539.2746590.

20 Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost-sharing: A simple approximation algorithm for the multicommodity rent-or-buy problem. In *44th Symposium on Foundations of Computer Science, FOCS 2003, Cambridge, MA, USA, October 11-14, 2003, Proceedings*, pages 606–615. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238233.

21 Bernhard Haeupler, D. Ellis Hershkowitz, and Thatchaphol Saranurak. Maximum length-constrained flows and disjoint paths: Distributed, deterministic, and fast. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1371–1383. ACM, 2023. doi:10.1145/3564246.3585202.

22 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 451–460. ACM, 2016. doi:10.1145/2933057.2933112.

23 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Near-optimal low-congestion shortcuts on bounded parameter graphs. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2016. doi:10.1007/978-3-662-53426-7_12.

24 Bernhard Haeupler, Taisuke Izumi, and Goran Zuzic. Low-congestion shortcuts without embedding. *Distributed Comput.*, 34(1):79–90, 2021. doi:10.1007/s00446-020-00383-2.

25 Bernhard Haeupler and Jason Li. Faster distributed shortest path approximations via shortcuts. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.33.

26 Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1325–1338. ACM, 2022. doi:10.1145/3519935.3520026.

27 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1166–1179. ACM, 2021. doi:10.1145/3406325.3451081.

28 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001. doi:10.1145/375827.375845.

29 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

30 Christoph Lenzen and Boaz Patt-Shamir. Improved distributed Steiner forest construction. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 262–271. ACM, 2014. doi:10.1145/2611462.2611464.

31 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. The point-to-point delivery and connection problems: complexity and algorithms. *Discret. Appl. Math.*, 36(3):267–292, 1992. doi:10.1016/0166-218X(92)90258-C.

32 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query, and streaming algorithms. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.

33 Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002. doi:10.1145/505241.505243.

34 Seth Pettie and Vijaya Ramachandran. A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. *SIAM J. Comput.*, 31(6):1879–1895, 2002. doi:10.1137/S0097539700371065.

35 Václav Rozhon, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected $(1+\varepsilon)$-shortest paths via minor-aggregates: Near-optimal deterministic parallel and distributed algorithms. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 478–487. ACM, 2022. doi:10.1145/3519935.3520074.

36 Stefan Voß. Steiner tree problems in telecommunications. In Mauricio G. C. Resende and Panos M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 459–492. Springer, 2006. doi:10.1007/978-0-387-30165-5_18.

37 Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based ℓ_1-oblivious routing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2549–2579. SIAM, 2022. doi:10.1137/1.9781611977073.100.

A Model-Agnostic Algorithm

A.1 Modified Centralized $(2+\varepsilon)$-Approximation

▶ **Theorem 8.** *For a graph $G=(V,E)$, edge costs $c\colon E\to\mathbb{N}$, and proper function f, the modified GW algorithm with incremental solution-set construction, $(1+\varepsilon')$-approximate distance computations, and $(1+\varepsilon'')$-deferred forest-function evaluation (Algorithm 2) yields a $(2-2/t)(1+\varepsilon')(1+\varepsilon'')^2$-approximation to the optimal solution, i.e., a $(2+\varepsilon)$-approximation for $\varepsilon',\varepsilon''\leq\varepsilon/4\leq 1/4$.*

Denote by $i=0,1,\dots$ the iterations of the while loop in Algorithm 2, calling each iteration a *merge phase*, and for each phase by (i) $r_i:=(1+\varepsilon'')^i\cdot\frac{\varepsilon''}{4}$ the radius r at the *beginning* of phase i (with $r_{-1}:=(1+\varepsilon'')^{-1}\cdot\frac{\varepsilon''}{4}$), (ii) U_i, c'_i F_i, F'_i, A_i, E_i, and $\mathcal{T}_i^1$ the values of the respective variables at the *end* of phase i, and (iii) $a_i:=|\mathcal{T}_i^1|$ the number of active components at the *end* of the phase i (with $a_{-1}:=t$). We prepare our proof in five lemmas.

▶ **Lemma 9.** *For all $i \in \mathbb{N}_0$ s.t. Algorithm 2 does not terminate at the end of phase i, we have (i) $r_i = (1+\varepsilon'')r_{i-1}$, (ii) $c'_{i+1}(e) \leq c'_i(e) \leq c(e)$ for each $e \in E_i$, (iii) U_i is spanned by $F'_i \cup A_i$, (iv) $U_i \subseteq U_{i+1}$, (v) $F'_i \cup A_i \subseteq F'_{i+1}$, (vi) $F_i \subseteq F_{i+1}$, (vii) $F_i \cup F'_i = F'_i \cup A'_i$ and (viii) (V, E_i) is connected and the weighted diameter w.r.t. reduced costs c' is decreasing.*

Proof. The first and the second statement hold by construction due to Lines 21 and 12, respectively. For the third statement, observe that F'_i connects each node in U_i to some node in $\mathcal{T}_i^1$, and the set M_i includes all edges that are both contained in U_i (i.e., whose reduced cost has become 0) and connect different connectivity components with respect to F'_i of U_i. Thus, the choice of A_i ensures that U_i is spanned by $F'_i \cup A_i$. This readily implies the forth statement, since $F'_i \cup A_i \subseteq E_i$, as well as the fifth, since all other edges e of reduced cost $c'(e) = 0$ are removed to obtain E_i, forcing any approximate SSSP solution to reselect the edges from $F'_i \cup A_i$ into F'_{i+1}. The sixth statement holds because the algorithm only adds edges to F, and the seventh statement holds by induction, using that $F_i = F_{i-1} \cup A_i \cup X$ for some set $X \subseteq F'_i$. Finally, for the eighth statement, observe that since U_i is spanned by $F'_i \cup A_i$, whose reduced cost is 0, and any edges not fully contained in U_i remain in E_i, the shortest path between any pair of nodes w.r.t. c' only becomes shorter: For any edge contained in U_i, there is now a path of reduced weight 0 between its endpoints, while any edge e with reduced cost $c'_i(e) > 0$ is still present and retains at most its original cost $c(e)$. ◀

▶ **Lemma 10.** *For $\varepsilon', \varepsilon'' \in n^{-\mathcal{O}(1)}$, Algorithm 2 terminates in $\mathcal{O}(\frac{\log n}{\varepsilon''})$ iterations of its loop.*

Proof. Since each of the operations within the loop terminate (assuming correct subroutines), it suffices to show the claimed bound on the number of loop iterations. To exit the while-loop, we must have $f(C) = 0$ for each $C \in \mathcal{C}$, where $\mathcal{C}$ is the set of connected components induced by our current edge set F. Recall that edge weights and hence the weighted diameter of the graph are polynomially bounded in n. As $\varepsilon'', \varepsilon' \in n^{-\mathcal{O}(1)}$, there is an $j \in \mathcal{O}(\frac{\log n}{\varepsilon''})$ for which $r_j = (1+\varepsilon'')^j \cdot \frac{\varepsilon''}{4}$ exceeds the weighted diameter of G times $1+\varepsilon'$. By Lemma 9 (viii), this entails that if we reach this phase, then U_j contains the entire connected graph (V, E_j). By Lemma 9 (iii), U_j (and thus V) is spanned by $F'_j \cup A_j$. The merge operation in Line 16 hence ensures that all terminals in $\mathcal{T}_{j-1}^1$ are connected by F_j. Denote by C the connectivity component of (V, F_j) containing the nodes in $\mathcal{T}_{j-1}^1$. For each terminal $\tau \notin \mathcal{T}_{j-1}^1 \setminus C$, τ lies in a connected component C' of (V, F_{j-1}) satisfying that $f(C') = 0$. This entails that we can partition $V \setminus C$ into two types of sets: (i) components C' of (V, F_{j-1}) satisfying that $f(C') = 0$, and (ii) non-terminals $v \in V \setminus \mathcal{T}$, which satisfy $f(\{v\}) = 0$ by definition. By disjointness of the proper forest function f, it follows that $f(V \setminus \mathcal{C}) = 0$, and by symmetry it follows that $f(C) = 0$. Finally, by Lemma 9 (vi), we have $F_{j-1} \subseteq F_j$. Therefore, each connectivity component C' of (V, F_j) other than C decomposes into connectivity components of (V, F_{j-1}), satisfying that $f(C') = 0$, and non-terminals, which again by disjointness implies that $f(C') = 0$. Thus, Algorithm 2 terminates by the end of phase $j \in \mathcal{O}(\frac{\log n}{\varepsilon''})$. ◀

▶ **Lemma 11.** *The edge set F output by Algorithm 2 is primal feasible.*

Proof. When Algorithm 2 terminates, we must have exited its while-loop, implying that we have $f(C) = 0$ for each $C \in \mathcal{C}$, where $\mathcal{C}$ is the set of connected components induced by our output set F. Consider any set $S \subseteq V$. If S non-trivially intersects a component $C \in \mathcal{C}$, i.e., $\emptyset \neq S \cap C \neq C$, then there is an edge of F in the cut defined by S, i.e., $|\delta(S) \cap F| \geq 1 \geq f(S)$. Otherwise, $S = \bigcup_{C \in \mathcal{C}_S} C$ for some $\mathcal{C}_S \subseteq \mathcal{C}$. As f satisfies disjointness and $f(C) = 0$ for each $C \in \mathcal{C}_S$, it follows that $|\delta(S) \cap F| \geq 0 = f(S)$ in this case, too. Thus, F is primal feasible. ◀

▶ **Lemma 12.** *The value LB output by Algorithm 2 is the cost of a feasible dual solution.*

Proof. Recall that $r = r_i = (1+\varepsilon'')^i \frac{\varepsilon''}{4}$ in phase i, and denote by j the index of the final phase Abbreviating $\Delta_i \coloneqq (1+\varepsilon')^{-1} r_i$, we can write the value LB as

$$LB = (1+\varepsilon')^{-1} \sum_{i=0}^{j} (1+\varepsilon'')^i \frac{\varepsilon''}{4} a_i = (1+\varepsilon')^{-1} \sum_{i=0}^{j} r_i a_i = \sum_{i=0}^{j} \Delta_i a_i \ .$$

Denote the cost reduction of an edge achieved in phase i as $\bar{c}_i(e) \coloneqq c_{U_i}(e) = c'_{i-1}(e) - c'_i(e)$, with $c'_{-1}(e) \coloneqq c(e)$, and the amount that y is increased in phase i for some set $S \subset V$ as $y_i(S)$. Now observe that to construct a feasible dual solution of value LB, it suffices to, in each phase $i \in [j]_0$, for each component C surviving phase i, increase the dual variables associated with the subsets $S \subset C$ in sum by Δ_i, while ensuring that for each $e \in E$, we maintain $\sum_{S\colon \delta(S) \ni e} y_i(S) \leq \bar{c}_i(e)$.

Since f is proper, we know that for each component C surviving phase i, there exists at least one component C' active in phase i such that $C' \subseteq C$. Therefore, to allocate Δ_i to dual variables associated with C, we can trace the construction of C from the set of components $\mathcal{C}' \coloneqq \{C' \subseteq C \mid f(C') = 1,\ C' \text{ is a connected component of } (V, F_{i-1})\}$ (where $F_{-1} \coloneqq \emptyset$) as follows. Starting with $\mathcal{C}'$ as it resulted from phase $i-1$, we increase the radii of all components $C' \in \mathcal{C}'$ at rate ρ and the y-variables of $C' \in \mathcal{C}'$ at rate $\frac{\rho}{1+\varepsilon'}$. Thus, we gradually construct U_i and reduce the costs of all edges in the affected cuts (i.e., increase $c_{U_i} = \bar{c}_i(e)$) until the first edge achieves $c'_i(e) = 0$ (or the phase ends). This happens *at the latest* when the first dual constraint in phase i becomes tight – it can happen earlier as the edge cost reductions associated with our radius increases are based on $(1+\varepsilon')$-approximate distances. When an edge achieves $c'_i(e) = 0$, we update F and $\mathcal{C}'$ to ensure that both endpoints of e lie in the same component, and we iterate the process described above.

Since C survived phase i, this process does not add an edge to F that lies in the cut $(C, V \setminus C)$ until we have increased the radius by r_i. We claim that at no point in this process, $\mathcal{C}'$ becomes empty. Assuming otherwise, we would have that $C = \bigcup_{C' \in \mathcal{C}'} C'$ with $f(C') = 0$ at the respective point of the process, implying the contradiction that $f(C) = 0$ by disjointness of f. Accordingly, the total increase of y-variables that we attribute to C during phase i is at least Δ_i, as desired. Moreover, our direct coupling of the y-variable increases with edge-cost reductions further ensures that the y-variables relevant for any individual edge e increase by at most $\bar{c}_i(e)$, i.e., its cost reduction in phase i, guaranteeing feasibility.

Summing over the a_i components surviving phase i, we increase the dual variables by at least $\Delta_i a_i$ per merge phase, concluding the proof. ◀

▶ **Lemma 13.** *Denoting by j the phase after which Algorithm 2 terminates, it holds that $c(F) \leq \left(2 - \frac{2}{t}\right)(1+\varepsilon'')^2 \sum_{i=0}^{j-1} r_i a_i$.*

Proof. Recall that merge phase $i \geq 0$ is the phase in which $r = r_i = (1+\varepsilon'')^i \cdot \frac{\varepsilon''}{4}$, and a_i is the number of active components *surviving* phase i. Moreover, note that all edges e added to F satisfy that $c'(e) = 0$ on termination, and that the cost reduction for each edge in phase i is at most $2r_i$, as each endpoint of the edge is contained in at most one tree of F'. Hence, the cost of each edge in F can be amortized over the phases i in which its cost is reduced by the algorithm, i.e., which it starts with $c'_{i-1}(e) > 0$, and in which it intersects U_i. Accordingly, F decomposes into a set of nested *shells* $U_i \setminus U_{i-1}$ (with $U_{-1} \coloneqq \emptyset$), which the algorithm iteratively and implicitly constructs around active components. This *shell-decomposition argument* is illustrated in Figure 2.

Since in phase i, the remaining edges $F \setminus F_{i-1}$ (where $F_{-1} \coloneqq \emptyset$) to be added to F form a forest with a_i active components as nodes, and the average degree of such a forest is at most $2 - 2/a_i$, we can bound the cost of the forest F output by Algorithm 2 from above as

$$c(F) \leq \sum_{i=0}^{j} r_i a_{i-1} \left(2 - \frac{2}{a_{i-1}}\right) \leq \left(2 - \frac{2}{t}\right)(1+\varepsilon'') \sum_{i=0}^{j} r_{i-1} a_{i-1} = \left(2 - \frac{2}{t}\right)(1+\varepsilon'') \sum_{i=-1}^{j-1} r_i a_i \,,$$

where $a_{-1} \coloneqq t$. Since F is a feasible primal solution by Lemma 11, each terminal must be incident with at least one edge from F, and the minimum edge weight is at least 1, we have that $c(F) \geq t/2$. On the other hand, $a_{-1} r_{-1} = \frac{\varepsilon'' t}{4(1+\varepsilon'')}$, yielding that
$c(F) \leq (1+\varepsilon'')c(F) - \frac{\varepsilon'' t}{2} = (1+\varepsilon'')(c(F) - 2a_{-1}r_{-1}) \leq \left(2 - \frac{2}{t}\right)(1+\varepsilon'')^2 \sum_{i=0}^{j-1} r_i a_i.$ ◀

Using the above lemmas, we can now prove Theorem 8.

Proof of Theorem 8. Assume that Algorithm 2 terminates at the end of phase j, i.e., $a_j = 0$; by Lemma 10, such a phase exists. Observe that the value LB output by Algorithm 2 then satisfies $LB = (1+\varepsilon')^{-1} \sum_{i=0}^{j} r_i a_i = (1+\varepsilon')^{-1} \sum_{i=0}^{j-1} r_i a_i$. By Lemma 11, F is a feasible primal solution. As LB is the cost of a feasible dual solution by Lemma 12, using Lemma 13, we obtain the desired approximation guarantee as

$$\frac{c(F)}{LB} \leq \frac{(2 - 2/t)(1+\varepsilon'')^2 \sum_{i=0}^{j-1} r_i a_i}{(1+\varepsilon')^{-1} \sum_{i=0}^{j-1} r_i a_i} = \left(2 - \frac{2}{t}\right)(1+\varepsilon')(1+\varepsilon'')^2 \,.$$ ◀

A.2 Specification of our Model-Agnostic Meta-Algorithm

The task of $(2+\varepsilon)$-approximating CFPs in any specific model reduces to simulating Algorithm 2. By Lemma 10, we can divide the computation into $\mathcal{O}(\frac{\log n}{\varepsilon''})$ phases, where, starting at phase 0, phase i grows components by radius $r_i = (1+\varepsilon'')^i \cdot \frac{\varepsilon''}{4}$. Each phase tackles six abstract problems, corresponding to the six building blocks of our algorithm (see Figure 1).

A.2.1 Problems Used as Building Blocks

▶ **Problem 3** (α-approximate Set-Source Shortest-Path Forest). *Given a connected graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{N}_0$ and a set of sources $S \subseteq V$, compute a forest F' spanning G such that for all nodes $v \in V$, $d_{F'}(S, v) \leq \alpha d(S, v)$, where $d(S, v) = \min_{u \in S}\{d(u, v)\}$, and $d_{F'}(u, v)$ is the weighted distance between u and v in F.*

Note that in phase i, we can confine ourselves to computing an α-approximate set-source shortest-path forest *up to distance r_i* (see Algorithm 2, Lines 9–10).

▶ **Problem 4** (Candidate-Merge Identification). *Given a graph $G = (V, E)$, edge costs $c\colon E \to \mathbb{N}_0$, and a rooted forest F' with a subset of its trees marked such that each node v in a marked tree knows that its tree is marked as well as the identity of its root τ_v, identify all edges $e = \{u, v\}$ that are in distinct marked trees and satisfy $c(e) = 0$.*

▶ **Problem 5** (Minimum Spanning Tree). *Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{N}_0$, compute the Minimum Spanning Tree of G.*

▶ **Problem 6** (Root-Path Selection). *Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{N}_0$, a rooted forest, and a set of marked nodes $S \subseteq V$, select the forest edges connecting each marked node to its root.*[12]

▶ **Problem 7** (Edge-Cost Reduction). *Given a graph $G = (V, E)$ with edge costs $c\colon E \to \mathbb{N}_0$, a radius r, and an output of Problem 3, compute, for each edge $e \in E$, $c'(e) = \max\left\{0, c(e) - \sum_{v \in e} \max\{r - d_{F'}(S, v), 0\}\right\}$.*

▶ **Problem 8** (Forest-Function Evaluation). *Given a graph $G = (V, E)$, a partition $\mathcal{C}$ of the node set such that each $C \in \mathcal{C}$ is connected, and a proper forest function $f\colon 2^V \to \{0, 1\}$, evaluate $f(C)$ for each component $C \in \mathcal{C}$.*

A.2.2 Meta-Algorithm Using the Building Blocks

Initialize F, $\mathcal{C}$, $\mathcal{T}^1$, c', and r as in Algorithm 2. Throughout our algorithm, we maintain a set of connected components $\mathcal{C}$ with activity statuses $f(C)$ for each $C \in \mathcal{C}$. At the beginning of phase 0, $\mathcal{C}$ contains exactly the singleton sets corresponding to all nodes, i.e., $\mathcal{C} = \{\{v\} \mid v \in V\}$, and the active components are the terminals. Each phase i of our algorithm (i.e., one loop iteration in Figure 1, simulating one while-loop iteration of Algorithm 2) then consists of the following steps, executed for $r = (1 + \varepsilon'')^i \cdot \frac{\varepsilon''}{4}$.

(1) *Approximate Set-Source Shortest-Path Forest (aSSSP).* Assign as temporary edge weight to $e \in E$ the reduced cost $c'(e)$ if $c'(e) > 0$ or $e \in F \cup F'$ (where $F' := \emptyset$ in phase 0). Compute a $(1 + \varepsilon')$-approximate (r-restricted) SSSP forest F', using the active terminals $\mathcal{T}^1 = \{\min\{v \mid v \in \mathcal{T} \cap C\} \mid C \in \mathcal{C}, f(C) = 1\}$ as sources, i.e., for each active component, $\mathcal{T}^1$ contains the terminal with the minimum identifier. After Step 1, for each node $v \in V \setminus \mathcal{T}^1$, we know its parent in the truncated SSSP forest, its closest source $u \in \mathcal{T}^1$ in the respective shortest-path tree (if any), and its distance $d_{F'}(u, v)$ to that source.

(2) *Edge-Cost Reduction (ECR).* Using the approximate r-restricted SSSP forest and the distances computed in Step 1, update the edge costs in accordance with Problem 7.[13]

(3) *Candidate-Merge Identification (CMI).* Using that nodes' parents and reduced edge costs are known, identify candidate merges M for adjacent trees of the aSSSP forest (Step 1).

(4) *Minimum Spanning Tree (MST).* Compute a Minimum Spanning Tree T of G with the following edge weights: (i) 0 for edges in F', i.e., the tree edges in the output of Step 1, (ii) 1 for edges in M, i.e., those determined in Step 3, and (iii) $+\infty$ (or a large value) for all other edges. Mark all selected edges of T that are also in the set M known from Step 3, i.e., the edges constituting A, and add them to F (thus *excluding* all edges with temporary weight greater than 1). For each connected component C' of the forest constituted by the selected edges of temporary weight 0 or 1 that contains a terminal $\tau \in C'$, set $\min\{v \in C' \mid f(\{v\}) = 1\}$ as the new identifier of the component C to be created from C', making it known to all $v \in C'$.

[12] Root-Path Selection can be reduced to *approximate transshipment* as follows: (i) count the number of marked nodes in each tree; (ii) set the demand of each marked node to -1 and the demand of the root of each tree to the number of marked nodes in the tree; (iii) set edge costs to 0 for tree edges and to $+\infty$ for all other edges; (iv) solve the approximate transshipment problem for these demands and edge weights; and (v) select all edges with non-zero flow in the output. Note that the only non-trivial step of the reduction is the computation of the demand, which boils down to a single Partwise Aggregation.

[13] We can keep the edge costs in $\mathbb{N}_0$ by making sure that phases end with integral values of r. Note that $\varepsilon' \geq n^{-\mathcal{O}(1)}$, or distance computations must be exact. Scale all weights by $\lceil 1/\varepsilon' \rceil$. Now rounding r up to the next integer has marginal impact on the approximation guarantee, as overgrowing by factor $(1 + \varepsilon')$ plus an additive 1 is not worse than overgrowing by factor $(1 + 2\varepsilon')$.

(5) *Root-Path Selection (RPS).* Connect the marked edges identified in Step 4 to the roots (i.e., the node with the same identifier as the component) of the components they connect by adding the necessary edges to F.

(6) *Forest-Function Evaluation (FFE).* Using the new component memberships known from Step 4, update the set $\mathcal{C}$ and evaluate $f(C)$ for each updated $C \in \mathcal{C}$. If $f(C) = 0$ for all such components, terminate and output F– else, continue with the next loop iteration.

A.3 Correctness and Complexity of our Model-Agnostic Meta-Algorithm

▶ **Theorem 14** (Model-Agnostic Shell-Decomposition Algorithm). *For $\varepsilon, \varepsilon', \varepsilon''$ as in Theorem 8, a graph $G = (V, E)$, edge costs $c\colon E \to \mathbb{N}$, and proper function f, the modular shell-decomposition algorithm (A.2.2) yields a feasible $(2+\varepsilon)$-approximation to the optimal solution.*

Proof. We prove the claim by induction on the phase i, going step by step through the algorithm given in Section A.2.2 and arguing why the computed objects, in particular F, match those of Algorithm 2. We augment the induction hypothesis by the claim that at the beginning of a phase, in Algorithm 2, E contains exactly the edges of non-zero reduced cost and $F_{i-1} \cup F'_{i-1}$. The induction anchor (phase $i = -1$) is given by the identical initialization of objects. For the step to phase $i \in \mathbb{N}_0$, observe first that by the induction hypothesis (in particular the additional claim), Step 1 computes the same F'_i and the same distances as Algorithm 2, and hence Step 2 yields the same c'_i. It follows that Step 3 computes the same set M of candidate merges, implying that Step 4 correctly determines A_i and adds it to F. Note that the latter step also updates component memberships and component identifiers, but does not yet evaluate whether $f(C) = 1$ for the new components. This is finally done in Step 6, such that in Step 1 of the next phase, the correct set $\mathcal{T}_i^1$ will be used. It remains to prove the additional claim that E_i contains exactly the edges of non-zero reduced cost and $F_{i-1} \cup F'_{i-1}$, which now is immediate from Line 17 and Lemma 9 (vii).

We conclude that both algorithms terminate at the end of the same phase j, returning the same forest $F = F_j$, which by Theorem 8 is a $(2 + \varepsilon)$-approximation. ◀

The proof of our main theorem now follows immediately.

Proof of Theorem 1. By Theorem 14, our modular shell-decomposition algorithm (Algorithm 2) delivers the desired approximation guarantee. Without loss of generality, we may assume that $\varepsilon \in n^{-\mathcal{O}(1)}$, as this is enough to enforce that ε times the cost of an optimal solution is smaller than 1, i.e., a 2-approximation is guaranteed. Thus, it is sufficient to instantiate the algorithm with $\varepsilon', \varepsilon'' \in n^{-\mathcal{O}(1)}$, such that by Lemma 10, the algorithm will terminate after $\mathcal{O}(\frac{\log n}{\varepsilon''}) = \mathcal{O}(\frac{\log n}{\varepsilon}) = \widetilde{\mathcal{O}}(\varepsilon^{-1})$ while-loop iterations. In each of these iterations (up to bookkeeping operations), aSSSP, MST, RPS, and FFE computations are performed exactly once, yielding the desired model-agnostic complexity. ◀

Towards Constant Time Multi-Call Rumor Spreading on Small-Set Expanders

Emilio Cruciani
European University of Rome, Italy

Sebastian Forster
University of Salzburg, Austria

Tijn de Vos
TU Graz, Austria

Abstract

We study a multi-call variant of the classic PUSH&PULL rumor spreading process where nodes can contact k of their neighbors instead of a single one during both PUSH and PULL operations. We show that rumor spreading can be made faster at the cost of an increased amount of communication between the nodes. As a motivating example, consider the process on a complete graph of n nodes: while the standard PUSH&PULL protocol takes $\Theta(\log n)$ rounds, we prove that our k-PUSH&PULL variant completes in $\Theta(\log_k n)$ rounds, with high probability.

We generalize this result in an expansion-sensitive way, as has been done for the classic PUSH&PULL protocol for different notions of expansion, e.g., conductance and vertex expansion. We consider small-set vertex expanders, graphs in which every sufficiently small subset of nodes has a large neighborhood, ensuring strong local connectivity. In particular, when the expansion parameter satisfies $\phi > 1$, these graphs have a diameter of $o(\log n)$, as opposed to other standard notions of expansion. Since the graph's diameter is a lower bound on the number of rounds required for rumor spreading, this makes small-set expanders particularly well-suited for fast information dissemination. We prove that k-PUSH&PULL takes $O(\log_\phi n \cdot \log_k n)$ rounds in these expanders, with high probability. We complement this with a simple lower bound of $\Omega(\log_\phi n + \log_k n)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Probabilistic algorithms

Keywords and phrases small set expansion, vertex expansion, rumor spreading, multi-call rumor spreading, push&pull protocol

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.26

Related Version *Full Version*: https://arxiv.org/abs/2508.18017

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702) and is supported by the Austrian Science Fund (FWF): P 32863-N and P 36280-N.

1 Introduction

Rumor spreading, which is loosely inspired by biological and social phenomena, is one of the most well-studied stochastic processes on graphs with a rich history in refined analyses [30, 48, 27, 38, 44, 28, 13, 11, 21, 52, 20, 22, 29, 34, 6, 32, 23, 25, 33, 41, 1, 46, 17, 31, 24]. In the classic *random phone call model* [19], an arbitrary node of the graph initially knows a piece of information, the rumor, which spreads across the graph in synchronous rounds until eventually all nodes are informed. In each round, every node calls one of its neighbors uniformly at random: in the PUSH protocol each informed node informs the node it calls, in the PULL protocol each uninformed node is informed by the node it called, and in the PUSH&PULL protocol both of these information exchanges happen simultaneously.


© Emilio Cruciani, Sebastian Forster, and Tijn de Vos;
licensed under Creative Commons License CC-BY 4.0



It is well known that PUSH, PULL, and PUSH&PULL on a *complete graph* with n nodes require $\Theta(\log n)$ rounds to complete rumor spreading with high probability[1] [30, 38]. Rumor spreading has been also extensively studied on *expander graphs*, with known tight bounds that relate the completion time of the process with different notions of expansion. In the following, we discuss only PUSH&PULL, due to strong lower bounds that exist for both PUSH and PULL in isolation [11, 52]. In particular, the protocol with high probability requires $\Theta(\varphi^{-1} \log n)$ rounds for graphs with *conductance* $\varphi \in [0, 1]$ [11], and $\Theta(\phi^{-1} \log n \log \Delta)$ rounds for graphs with maximum degree Δ and *vertex-expansion* $\phi \in [0, 1]$ [34, 32]. It is also known that PUSH&PULL requires $\Omega(\sqrt{n})$ rounds with high probability in graphs with constant *edge-expansion* [13]. The study of rumor spreading protocols on expanders, subsequently, has been used as a building block for designing distributed information dissemination algorithms [9, 10, 8, 35], some of which are based on expander decompositions [54].

Faster rumor spreading can be achieved on specific topologies or by modifying the process. On power-law degree distributed graphs, the process completes in $O(\log \log n)$ rounds if the exponent of the power-law β is between 2 and 3, while it needs $\Omega(\log n)$ rounds if β is greater than 3 [29]. A variant of the process, where nodes can use direct addressing, i.e., are able to contact neighbors whose address was learned before, takes $\Theta(\log \log n)$ rounds [36, 6].

With the goal of making rumor spreading even faster, in this paper we study a *multi-call variant* of the classical protocol. We call this the k-PUSH&PULL protocol, where in each round every informed node samples k neighbors and sends the rumor to them, while every uninformed node samples k neighbors and requests the rumor from them. The sampling happens uniformly at random and with replacement in both PUSH and PULL operations.

This process naturally interpolates between classical single-call rumor spreading and full broadcasting. It involves random selective communication, as in the former, while nodes simultaneously interact with multiple neighbors, as in the latter. This interpolation intuitively results in a trade-off between the amount of communication and the spreading time of the two processes. Classical rumor spreading requires minimal communication, while broadcasting is optimal in spreading time, matching the graph's diameter in the worst case.

Somewhat similar processes have been studied for the special case $k = 4$ on random regular graphs [7], with a randomized number of simultaneous calls on the complete graph [47, 24], and in the asynchronous setting considering multiple pulls on the complete graph [42, 50]. For the related random-walk process, a multi-walk extension similar in spirit to our multi-call extension for rumor spreading has been studied by Alon et al. [2]. Apart from these works we are not aware of any paper studying the multi-call setting explicitly. While the analogy to a biological or social process might not be given anymore for the multi-call variant, we still believe that increasing the number of simultaneous calls is a quite natural generalization that captures a relevant aspect of information dissemination under bandwidth constraints. Furthermore, note that the classic "single-call" variant allows each node to *receive* multiple calls at the same time.[2] Therefore, it seems natural to also allow nodes to *initiate* several calls at the same time.

This leads to the central question of our work:

Question: *How much communication is required for sub-logarithmic time rumor spreading?*

[1] We say that a statement holds *with high probability* (w.h.p. for short) if it holds with probability at least $1 - n^{-c}$, where c is a positive constant hidden in the asymptotic notation.

[2] Restricted variants in which nodes receiving multiple calls can spread the rumor only to a single neighbor [17] cannot achieve performance guarantees comparable to the unrestricted variant in conductance expanders and vertex expanders [31].

1.1 Our Contribution

The starting point of our investigation is the study of multi-call rumor spreading on the *complete graph*. Using standard arguments that we present as a warm-up in Section B, we show that the k-PUSH&PULL protocol requires $\Theta(\log_k n)$ rounds with high probability. This gives an interesting trade-off between the classic single-call bound of $\Theta(\log n)$ and a constant number of rounds for a polynomial number of simultaneous calls.

Next, we extend our question to *expander graphs*. Our goal is to generalize the trade-off that we get for k-PUSH&PULL on the complete graph to incorporate an expansion parameter ϕ, similarly to the works mentioned above for the single-call model. While this is an intriguing combinatorial question in its own right, we believe that the recent use of expanders in graph neural networks [18, 53] adds additional motivation for revisiting the foundations of information dissemination in expanders.

Our focus is on *small-set expanders*, that have initially been studied for the notion of conductance (see e.g., [49, 3]), and later for edge expansion (see e.g., [43]) and vertex expansion (see e.g., [16, 39, 14]). Small-set vertex expansion has also been identified as a crucial property of networks for designing routing protocols [4]. A non-negligible part of our contribution consists of studying properties of *small-set vertex expanders* in a particular regime where the vertex expansion ϕ is larger than 1. In particular, we establish upper and lower bounds on the *diameter* of such graphs (Section C), which, unlike other standard notions, is *sub-logarithmic*.

This property is crucial for achieving sub-logarithmic rumor spreading time, as the graph diameter provides a fundamental lower bound on the number of rounds required. Roughly speaking, our main result (see Section 2 and Theorem 2) implies that, with high probability, a constant number of rounds suffices for multi-call rumor spreading in a wide regime of small-set vertex expanders when both the number of calls k and the expansion parameter ϕ are polynomial in n. We also give a complementary lower bound (see Theorem 3) showing that both parameters need to be polynomial to achieve a constant number of rounds.

1.2 Formal Statement of Our Results

Small-set vertex expanders

Let $G = (V, E)$ be an unweighted graph. For every $S \subseteq V$, we write $\overline{S} := V \setminus S$ for the complement of S. We also define the neighborhood of a set S as $N(S) := \{v \in V : \exists s \in S, \{s, v\} \in E\}$ and its boundary $\partial S := N(S) \cap \overline{S}$. For a single node v, we write $N(v) := N(\{v\})$. Note that $v \notin N(v)$, but $N(S) \cap S$ can be nonempty for larger S. Using this notation, we are ready to define the class of *small-set vertex expanders* (see, e.g., [16, 39, 14]).

▶ **Definition 1.** *Let $\phi \in (0, n)$ and $\alpha \in (0, \frac{1}{2}]$. We say a graph G with n nodes is a* (ϕ, α)-(vertex) expander *if*

$$\min_{\substack{S \subseteq V \text{ s.t.} \\ 0 < |S| \leq \alpha n}} \frac{|\partial S|}{|S|} \geq \phi.$$

The name small-set comes from the fact that the expansion property holds for sets of size at most αn, opposed to the classical definition where $\alpha = \frac{1}{2}$. The regime $\alpha < \frac{1}{2}$ is conceptually different and has implications on the expansion parameter ϕ, allowing $\phi > 1$. When $\alpha = \frac{1}{2}$, in fact, we *cannot* have $\phi > 1$. Indeed, any set S with $|S| = n/2$ nodes satisfies $\frac{|\partial S|}{|S|} \leq \frac{|V \setminus S|}{|S|} = 1$. More generally, there are no (ϕ, α)-expanders for $\alpha > \frac{1}{1+\phi}$ (see Lemma 18).

However, if we take $\alpha = \frac{1}{1+\phi}$, we only get a very restricted class of graphs: such (ϕ, α)-expanders have extremely low diameter and are very dense; they have diameter 2 and minimum degree $\Theta(n)$ (see Lemma 19). On the other hand, if we take α too small, the expansion property becomes local and the graph is no longer necessarily connected. To be precise, for $\alpha \leq \frac{1}{2+2\phi}$, there exist (ϕ, α)-expanders that are disconnected (see Lemma 20). In between these two bounds, we get a guarantee on the diameter that depends on ϕ. For $\frac{1}{2+2\phi} < \alpha \leq \frac{1}{1+\phi}$, we have that (ϕ, α)-expanders have diameter at most $O(\log_\phi n)$ (see Lemma 21). We note that for $\phi = \omega(1)$ the diameter is $o(\log n)$ and that for polynomial ϕ it becomes constant.

Next, we provide two examples of vertex expanders where the bound on the diameter is tight. First of all, we obtain that complete graphs are (ϕ, α)-vertex expanders for $0 < \phi \leq n-1$ and $\alpha \leq \frac{1}{1+\phi}$ (see Lemma 22). Secondly, we show that random graphs are vertex expanders. More precisely, we show that Erdős-Rényi random graphs, where between each pair of nodes there is an edge with probability $p = \frac{3\phi}{n}$, are (ϕ, α)-vertex expanders for $\alpha \leq \frac{1}{1+1.6\phi}$ (see Lemma 23). We note that in this regime of p these graphs have diameter $\Theta(\log_\phi n)$ with high probability [15].

We also note that concurrent work by Hsieh et al. [37] gives an explicit $\Theta(\phi)$-regular construction for small-set vertex expanders.

Rumor Spreading

For $\frac{1}{2+2\phi} < \alpha \leq \frac{1}{1+\phi}$, we study rumor spreading on (ϕ, α)-vertex expanders through the k-PUSH&PULL protocol.

▶ **Theorem 2.** *Let $\phi > 1$, $\alpha > \frac{1}{2+2\phi}$, let $G = (V, E)$ be a (ϕ, α)-expander, and let $k > \log^3 n$. Then w.h.p. rumor spreading with the k-PUSH&PULL protocol requires the following number of rounds:*

$$O\left(\left(\log_\phi n + \phi^{-1}\left(\alpha - \tfrac{1}{2+2\phi}\right)^{-1}\right)\log_k n\right).$$

Before commenting on this round complexity, we give a lower bound that follows immediately from the fact that on complete graphs we need $\Omega(\log_k n)$ rounds (see Lemma 17) and on Erdős-Rényi random graphs (which are (ϕ, α)-expanders by Lemma 23) we need $\Omega(\log_\phi n)$ rounds, since they have diameter $\Omega(\log_\phi n)$ [15].

▶ **Theorem 3.** *Let $n \geq 1$, $\phi > 1$, $\alpha \leq \frac{1}{1+1.6\phi}$ and $k \geq 2$. Then there exist a (ϕ, α)-expander $G = (V, E)$ on n nodes such that w.h.p. rumor spreading with the k-PUSH&PULL protocol requires $\Omega(\log_\phi n + \log_k n)$ rounds.*

We make the following observations about the round complexity we get in Theorem 2:

- The round complexity goes to infinity as α approaches $\frac{1}{2+2\phi}$, where expanders can be disconnected and hence rumor spreading never finishes. Interestingly, an analogous term appears in the analysis of the classic protocol on random graphs, making the rumor spreading time go to infinity in the non-connected regime [46].
- For $\alpha \geq \frac{1}{2+2\phi} + \Omega\left(\frac{1}{\phi \log_\phi n}\right)$ the number of rounds simplifies to $O(\log_k n \log_\phi n)$.
- If additionally $\phi = n^{\Omega(1)}$, then we obtain $O(\log_k n)$, which is optimal due to Theorem 3.
- If instead $k = n^{\Omega(1)}$, then we obtain $O(\log_\phi n)$, which is optimal due to Theorem 3.

We believe that our work opens several interesting research directions for the community. Closing the gap between our upper and lower bound is left as an open problem. Extending the proof technique of the state of the art analysis [32] to the multi-call setting in our opinion

appears to be non trivial; it is also not clear if this would be sufficient to actually close the gap. Another direction is to explore the use of such expanders in applications beyond rumor spreading. They can serve as a natural alternative to traditionally used low-diameter graphs, with the benefit of having combinatorial expansion properties.

Lower bounds

The fact that small-set vertex expanders allow $\phi > 1$ and hence sub-logarithmic diameter is fundamental to achieve our results. We observe that the classical notions of expansions considered in the literature do not have these properties and, therefore, the multi-call variant cannot speed up the rumor spreading time on these graphs. The most studied notions of expanders for which these properties do not hold are the following:

1. *ϕ-conductance expanders*, where for $\phi \in (0,1)$ we have

$$\min_{\substack{S \subseteq V \text{ s.t.} \\ 0 < \mathrm{Vol}(S) \leq m/2}} \frac{|E(S,\overline{S})|}{\mathrm{Vol}(S)} \geq \phi,$$

   where $|E(S,\overline{S})|$ is the number of edges crossing the cut $(S,\overline{S})$, and $\mathrm{Vol}(S) := \sum_{u \in S} \deg(u)$ is the volume of S, with $\deg(u)$ being the degree of a node $u \in V$.
2. *ϕ-edge expanders*, where for $\phi \in (0,n)$ we have

$$\min_{\substack{S \subseteq V \text{ s.t.} \\ 0 < |S| \leq n/2}} \frac{|E(S,\overline{S})|}{|S|} \geq \phi.$$

3. *ϕ-vertex expanders*, where for $\phi \in (0,1)$ we have

$$\min_{\substack{S \subseteq V \text{ s.t.} \\ 0 < |S| \leq n/2}} \frac{|\partial S|}{|S|} \geq \phi.$$

Regarding conductance-expanders, the following statement holds.

▶ **Lemma 4.** *Let $n \geq 1$, $\phi \in (0,1)$ and $k \geq 1$. There exists a ϕ-conductance expander on n nodes such that k-PUSH&PULL takes $\Theta(\phi^{-1} \log n)$ rounds.*

The proof follows since the same upper bound holds already for $k = 1$, and the lower bound comes from the diameter of ϕ-conductance expanders [11].

Rumor spreading on ϕ-edge expanders on the other hand, is not very appealing in the first place due to the strong lower bound of $\Omega(\sqrt{n})$ rounds for the single-call variant – which even holds in edge expanders with low diameter [13].

Regarding vertex expanders, we have the following statement.

▶ **Lemma 5.** *Let $n \geq 1$, $\phi \in (0,1)$ and $k \geq 1$. There exists a ϕ-vertex expander on n nodes such that k-PUSH&PULL takes $\Omega(\phi^{-1} \log n)$ rounds.*

The lemma follows from a lower bound on the diameter of ϕ-vertex expanders [34]. A tight bound of $\Theta(\phi^{-1} \log n \log \Delta)$ is known for the single-call variant when $\phi \leq 1$ [34, 32]. This means that the potential for savings in the multi-call variant is limited to a single $\log \Delta$-factor compared to the single-call variant; eliminating this factor is, to the best of our judgment, a quite fine-grained endeavor that we deliberately omit from this paper with its more conceptual focus.

Another class of graphs with very good expansion properties is that of *Ramanujan graphs*, whose spectral gap in the matrix representation of the graph is almost as large as possible. In particular, they can achieve sublogarithmic diameter, but only under very restrictive conditions, i.e., when they are $\Theta(n)$-regular [51]. In contrast, our class of small-set vertex expanders allows sublogarithmic diameter while being significantly sparser.

1.3 Technical Challenges

The proof we provide for rumor spreading in vertex expanders with $\phi > 1$ is based on the ideas for the case of expansion at most 1, by Giakkoupis and Sauerwald [34]. There are three complications we need to overcome. Let I_t denote the informed nodes in round t. We will show that in any round either I_t grows significantly or the boundary ∂I_t grows significantly. While this growth is a constant factor for the case $\phi < 1$, leading to a $\log n$ in the round complexity, we need to grow more aggressively. Intuitively, if the expected growth in one round is μ, then the expected growth in a round with k-parallel calls is $k\mu$. However, this brings us to the first issue.

Overlap of Parallel Calls

To some extent this challenge already appears in the `PUSH` model: the fact that two nodes individually push the rumor to a neighbor does not mean two new nodes are informed; they can push the rumor to the same node. This phenomenon is exacerbated when we call k times. Moreover, it now also impacts pulling: if one node has two successful parallel pulls in a round, then this does not contribute as two nodes pulling. In other words, we cannot simply sum the expected gain from separate pushes and pulls to obtain the expected number of newly informed nodes. Let us consider these probabilities to understand this situation better.

Let $u \in \partial I_t$ be an uninformed node. The probability that u pulls the rumor from I_t in one round of k-`PUSH&PULL` is

$$1 - \left(1 - \frac{|N(u) \cap I_t|}{\deg(u)}\right)^k = p.$$

Now if $\deg(u) \geq k|N(u) \cap I_t|$, we can use a binomial expansion to lower bound this probability by $\frac{k|N(u)\cap I_t|}{2\deg(u)}$, which is a factor $k/2$ higher than the probability that a single call succeeds. For nodes with lower degree we note that $p \geq 1/2$, so we can still say that they get informed with good probability. However, this does not carry over a factor k compared to the single-call situation. We are able to show a sufficient growth by carefully analyzing the number of nodes with high and low degrees and using different arguments for each case.

In this brief description we have only highlighted the issue. The actual solution is more involved but based on this intuition. For details see Section 3.2 and Section 3.3.

Probabilistic Guarantees

The second issue we need to deal with is the probability of successfully growing I_t or ∂I_t through `k-PUSH&PULL`. Intuitively, there are two sides to what is going on here: on the negative side, we have only a sub-logarithmic number of rounds, which is often not great to give bounds with high probability. On the positive side, we have significant growth in each step, which is good for tail bounds.

From this second observation, we see that in some cases a Chernoff bound suffices. However, in some situations, we cannot use a simple Chernoff bound since the random variables are neither independent nor negatively correlated. In particular, this happens when we want to show that the boundary ∂I_t grows. Let $u \in \partial I_t$ be a node in the boundary of I_t and $v_1, v_2 \in N(u) \setminus I_t$ be two uninformed neighbors of u. The events "$v_1 \in \partial I_{t+1}$" and "$v_2 \in \partial I_{t+1}$" are not independent nor negatively associated as both may occur as a consequence of the event "$u \in I_{t+1}$". In some cases we can use the bounded difference inequality, see Section A, that can give tail bounds on *functions* of independent random variables which we design to be equal to the sum of the correlated ones discussed above. See for example Section 3.2 and Section 3.3.

In other cases, the function that measures progress does not satisfy the conditions for the bounded difference inequality. Here we use an additional trick: we do not use the full potential of k parallel calls, but only $k/\log n$ calls. Each batch of $k/\log n$ call succeeds with constant probability, and since we have $\log n$ of these independent batches in parallel, at least one succeeds with high probability. This comes at the drawback that we only grow a factor $k/\log n$. By our assumption that $k > \log^3 n$, we get $k/\log n > k^{2/3}$, which means that $\log_{k/\log n} n = O(\log_k n)$.

No Expansion for Large Sets

The last complication we point out here is that small-set expansion is, by definition, only guaranteed for sets S of size at most αn. Therefore, we cannot use the expansion property to argue that I_t keeps growing until it hits $n/2$ nodes – at which point a standard symmetry argument shows that the process completes within a factor 2 in the round complexity. To handle the case where more than αn nodes are informed, we show that the size of the boundary ∂I_t is at least a constant fraction of n, meaning that if the set of informed nodes I_t keeps growing by a constant fraction of the boundary ∂I_t, the process still completes in a constant number of additional iterations. To be precise, this leads to the factor $\phi^{-1}(\alpha - \frac{1}{2+2\phi})^{-1}$ in our round complexity.

2 Multi-Call Rumor Spreading on Small-Set Vertex Expanders

The goal of this section is to prove the following theorem, which extends the study of multi-call rumor spreading on small-set vertex expanders with expansion $\phi > 1$ (see Definition 1).

▶ **Theorem 2.** *Let $\phi > 1$, $\alpha > \frac{1}{2+2\phi}$, let $G = (V, E)$ be a (ϕ, α)-expander, and let $k > \log^3 n$. Then w.h.p. rumor spreading with the k-PUSH&PULL protocol requires the following number of rounds:*

$$O\left(\left(\log_\phi n + \phi^{-1}\left(\alpha - \tfrac{1}{2+2\phi}\right)^{-1}\right)\log_k n\right).$$

The proof of this theorem is based on the ideas for the case of expansion at most 1, by Giakkoupis and Sauerwald [34]. Where applicable, we follow their notation.

Let I_t denote the set of informed nodes in round t. Our intermediate goal is to show that either I_t grows or ∂I_t grows. We do this by partitioning the nodes in the boundary according to their degree, and then analyzing each set separately. More formally, we partition the nodes in the boundary ∂I_t into different sets A_i defined as

$$A_i := \{u \in \partial I_t : d_i \le \deg(u) < 2d_i\}, \text{ with } d_i := 2^{i-1}.$$

Note that there are at most $\log n$ of such sets. We consider only those A_i's that are sufficiently big and such that the degree of the nodes in the set is bounded by quantities depending on the size of the set itself or is large with respect to the size of the boundary ∂I_t. Formally, let

$$\mathcal{I} := \left\{ i : |A_i| \ge \frac{|\partial I_t|}{4\log n} \wedge \left(d_i \le 16\,|A_i| \vee d_i \ge 2\,|\partial I_t|\right)\right\} \tag{1}$$

be the set of indices of the A_i's we take into account. We note that by considering such sets only, we consider at least half of the nodes in the boundary ∂I_t. We include a proof for completeness.

▶ **Lemma 6** ([34]). *It holds that $\left|\bigcup_{i\in\mathcal{I}} A_i\right| \ge |\partial I_t|/2$.*

Proof. The worst case with respect to the condition $|A_i| \geq \frac{|\partial I_t|}{4 \log n}$ is that all but one of the A_i's have size $|A_i| = \lceil |\partial I_t|/4 \log n \rceil - 1 < |\partial I_t|/4 \log n$. The worst case with respect to the condition $d_i \leq 16\,|A_i| \vee d_i \geq 2\,|\partial I_t|$ is that $|A_i| = \lceil d_i/16 \rceil - 1 \leq d_i/16$ and $d_i < 2|\partial I_t|$, for all $i < \log(2|\partial I_t|)$. Hence, since the A_i's are disjoint, we have

$$\left| \bigcup_{i \notin \mathcal{I}} A_i \right| = \sum_{i \notin \mathcal{I}} |A_i| \leq \frac{|\partial I_t|}{4 \log n} \log n + \frac{1}{16} \left(2|\partial I_t| + |\partial I_t| + |\partial I_t|/2 + |\partial I_t|/4 + \ldots + 2\right) \leq \frac{|\partial I_t|}{2}$$

which implies the thesis. ◀

In Section 3 we show the following lemma, which is the main technical lemma used in the proof of Theorem 2.

▶ **Lemma 7.** *For each $i \in \mathcal{I}$ and at every round t, we have that with high probability either at least $\frac{|A_i|}{128}$ nodes from A_i are informed within $r = O(\log_k n)$ rounds, or the boundary ∂I_t grows by at least $\frac{k^{1/6}}{64}|\partial I_t|$ within $O(1)$ rounds.*

Lemmas 6 and 7 lead to the following claim.

▶ **Lemma 8.** *If $|I_t| \leq \frac{n}{2}$, for $r = O(\log_k n)$ we have that with high probability*

$$|I_{t+r} \setminus I_t| \geq \frac{1}{256}|\partial I_t|.$$

Proof. First, we show that in each phase of r' rounds with high probability we are in either of the following cases:

a) $|I_{t+r'} \setminus I_t| \geq \frac{1}{256}|\partial I_t|$, for $r' = O(\log_k n)$,

b) $|\partial I_{t+r'} \setminus \partial I_t| \geq \frac{k^{1/6}}{64}|\partial I_t|$, for $r' = O(1)$.

Lemma 7 guarantees that for each $i \in \mathcal{I}$ with high probability either $\frac{|A_i|}{128}$ nodes from A_i are informed within $r = O(\log_k n)$ rounds, or the boundary ∂I_t grows by at least $\frac{k^{1/6}}{64}|\partial I_t|$ within $r = O(1)$ rounds.

The thesis follows by considering two cases. If there is $i \in \mathcal{I}$ such that the boundary grows then the thesis follows immediately. Otherwise, it must hold that for all $i \in \mathcal{I}$ there are at least $\frac{|A_i|}{128}$ nodes in A_i which get informed. Since the A_i's are disjoint, by Lemma 6 we conclude that

$$|I_{t+r} \setminus I_t| \geq \sum_{i \in \mathcal{I}} \frac{|A_i|}{128} = \frac{1}{128} \left| \bigcup_{i \in \mathcal{I}} A_i \right| \geq \frac{|\partial I_t|}{256}.$$

Next, we notice that Case b can only occur $O(\log_{\frac{k^{1/6}}{64}} n) = O(\log_k n)$ consecutive times before we cover the entire graph. Hence after $r = O(\log_k n) + O(\log_k n) = O(\log_k n)$ rounds, Case a holds. ◀

Next, we use this lemma to show that we reach more than half of the graph. We show that in $O(\log_\phi n)$ phases of $O(\log_k n)$ rounds we reach αn nodes. Let $I_{(t)}$ denote the set of informed nodes in phase t.[3] We start by showing by induction that until $|I_{(t)}| \geq \alpha n$, we have $|\partial I_{(t)}| \geq (\frac{\phi}{256})^t$.

[3] Note that we use I_t for the set of informed nodes in *round* t.

- **Base case.** By expansion, every node $v \in V$ has $\deg(v) = |\partial\{v\}| \geq \phi|\{v\}| = \phi$. So we have that $|\partial I_{(1)}| \geq \phi$.
- **Induction step.** Suppose that $|\partial I_{(t)}| \geq (\frac{\phi}{256})^t$. By Lemma 8 have $|I_{(t+1)}| \geq \frac{1}{256}|\partial I_{(t)}|$. So suppose $|I_{(t+1)}| \leq \alpha n$, then we have $|\partial I_{(t+1)}| \geq \phi|I_{(t+1)}|$ by expansion of $I_{(t+1)}$. We obtain $|\partial I_{(t+1)}| \geq \frac{\phi}{256}|\partial I_{(t)}| \geq \frac{\phi}{256}(\frac{\phi}{256})^t \geq (\frac{\phi}{256})^{t+1}$, using the induction hypothesis.

So we conclude that in $O(\log_{\phi/256}(\alpha n)) = O(\log_\phi n)$ phases of $O(\log_k n)$ rounds we cover αn nodes, using in total $O(\log_\phi n \cdot \log_k n)$ rounds.

Now suppose that $\alpha n < |I_{(t)}| \leq \frac{n}{2}$. Let $S \subseteq I_{(t)}$ be any subset of size exactly $|S| = \alpha n$, so $|I_{(t)} \setminus S| \leq \frac{n}{2} - \alpha n$. By expansion of S, we have $|\partial S| \geq \phi|S| = \phi\alpha n$. We now see that

$$|\partial I_{(t)}| \geq |\partial S \setminus I_{(t)}| = |\partial S \setminus (I_{(t)} \setminus S)| \geq |\partial S| - |I_{(t)} \setminus S| \geq \phi\alpha n - (\tfrac{n}{2} - \alpha n) = (\phi+1)\alpha n - n/2.$$

By Lemma 8 we now have to have that in each phase of $O(\log_k n)$ rounds, $I_{(t)}$ grows by

$$\frac{1}{256}|\partial I_{(t)}| \geq \frac{1}{256}((\phi+1)\alpha n - n/2).$$

So we cover more than $n/2$ nodes in the following number of phases

$$\frac{n/2}{\frac{1}{256}((\phi+1)\alpha n - n/2)} = O\left(\frac{1}{(\phi+1)\alpha - 1/2}\right) = O\left(\frac{1}{\phi(\alpha - \frac{1}{2+2\phi})}\right).$$

As each phase last $O(\log_k n)$ rounds, in total, we see that the rumor spreads to more than $n/2$ nodes in the following number of rounds

$$R = O\left(\left(\log_\phi n + \frac{1}{\phi\left(\alpha - \frac{1}{2+2\phi}\right)}\right)\log_k n\right).$$

Now Lemma 15 states that the number of rounds we need to spread the rumor from I_t to a node $u \in V \setminus I_t$ is the same as the number of rounds we would need to spread the rumor from u to anywhere in I_t. Since we have shown that we reach more than half of the graph in R rounds, and that $|I_t| > n/2$, a rumor started at u reaches I_t in another R rounds. We conclude that the spreading process finishes after $2R$ rounds in total, proving Theorem 2.

3 Proof of Lemma 7

We look at each set A_i for $i \in \mathcal{I}$ separately, and we upper bound the number of rounds until either a large fraction of nodes in A_i gets informed or the size of the boundary increases by a large factor as a result of nodes in A_i being informed. We distinguish three cases informally involving sets of nodes with low, medium, and high degree. We define these cases according to the following conditions:

1. Low degree: $d_i \leq 1024 \cdot 96 \cdot k$.
2. Medium degree: $1024 \cdot 96 \cdot k < d_i \leq 16\,|A_i|$.
3. High degree: $d_i \geq 2\,|\partial I_t|$.

Note that the three cases do not cover all nodes in the boundary ∂I_t. However, they do cover all nodes in the $|A_i|$'s for $i \in \mathcal{I}$. This is important since, due to Lemma 6, it implies that we consider at least half of the nodes in ∂I_t.

We analyze the above three cases separately in the three following subsections, that will be further subdivided into a total of 7 cases: in 4 cases I_t grows, while in 3 cases ∂I_t grows, always with high probability. By summarizing all of them we get with high probability that either

$$|I_{t+r} \setminus I_t| \geq \min\left\{\frac{1}{4}, \frac{1}{20}, \frac{1}{128}, \frac{1}{8}\right\}|A_i| = \frac{|A_i|}{128}$$

within $r = O(\max\{1, \log_k n, 1, 1)\} = O(\log_k n)$ rounds or

$$|\partial I_{t+r} \setminus \partial I_t| \geq \min\left\{\frac{k}{32\log n}, \frac{\sqrt{k}}{4}, \frac{k}{64}\right\}|A_i| \geq \frac{k^{1/6}}{64}|\partial I_t|$$

within $r = O(1)$ rounds, since by definition $|A_i| \geq \frac{|\partial I_t|}{4\log n}$ and by assumption $k > \log^3 n$.

3.1 Sets with low degree nodes

Recall that in this case every $u \in A_i$ is such that

$$d_i \leq 1024 \cdot 96 \cdot k.$$

Since the degree of these boundary nodes is low, it is likely that many of them will directly pull the rumor. We prove a more general lemma that will be used in other cases as well and formalizes the fact that nodes that are "well-connected" to I_t are likely to pull the rumor directly. In particular we consider nodes of a generic subset $B \subseteq \partial I_t$ of the boundary of I_t and bound the number of rounds needed by a constant fraction of nodes in B to pull the rumor from I_t, with high probability.

▶ **Lemma 9.** *Let $B \subseteq \partial I_t$ be such that $|B| = \Omega(\log n)$. Let $0 < q \leq 1$ and suppose that for every node $u \in B$ at least a q-fraction of its neighbors is in I_t. Then with high probability at least $\frac{|B|}{4}$ nodes of B have pulled the rumor from I_t in $\lceil \frac{1}{qk} \rceil$ rounds.*

Proof. We consider two different cases, depending on the value of q. We show for both cases that the probability of pulling from an informed node is at least $1/2$.
Case 1. If $qk > 1$, we show we only need 1 round. The probability that any $u \in B$ pulls from I_t in one round is $1 - \left(1 - \frac{|N(u) \cap I_t|}{\deg(u)}\right)^k \geq 1 - (1-q)^k$. Since $qk > 1$, then $1 - (1-q)^k \geq 1/2$.
Case 2. If $qk \leq 1$, then the probability that a node pulls the rumor in $\lceil \frac{1}{qk} \rceil$ rounds is $1 - (1-q)^{k \cdot \frac{1}{qk}} \geq 1 - e^{-1} \geq 1/2$.

This means that the expected number of nodes that pull in this many rounds is at least $|B|/2$. Since these pulls are independent random variables and $|B| = \Omega(\log n)$, a Chernoff bound gives us that at least $|B|/4$ nodes are informed with high probability. ◀

We are now ready to show that the set A_i grows sufficiently. For each node $u \in A_i$ the fraction of neighbors that u has in I_t is at least $1/\deg(u) \geq 1/(2d_i)$. Then Lemma 9 gives that with high probability a fraction $1/4$ of the nodes in A_i pull the rumor from I_t in at most $r = \lceil 2 \cdot 1024 \cdot 96 \cdot k/k \rceil = O(1)$ rounds. By definition of $\mathcal{I}$ in Equation (1), with high probability it follows that

$$|I_{t+r} \setminus I_t| \geq \tfrac{1}{4}|A_i|.$$

3.2 Sets with medium degree nodes

Recall that in this case every $u \in A_i$ is such that

$$1024 \cdot 96 \cdot k < d_i \leq 16\,|A_i|.$$

Unlike the previous case, nodes with medium degree can either contribute directly to the growth of I_t or to the growth of ∂I_t. To describe these different types of contributions, for each node $v \in V$ let us denote the number of neighbors of v in A_i as

$$h_i(v) := |N(v) \cap A_i|.$$

Moreover, let us define the set of nodes that are neither informed or in the boundary of the informed nodes as

$$S_t := V \setminus (I_t \cup \partial I_t).$$

We further distinguish three cases, depending on the *volume* of A_i, we define the volume of a set S as $\mathrm{Vol}(S) := \sum_{v \in S} \deg(v)$.

- **i)** It holds that $\sum_{v \in S_t \text{ s.t.} h_i(v) \leq d_i \log n/k} h_i(v) \geq \frac{1}{2}\mathrm{Vol}(A_i)$.
- **ii)** It holds that $\sum_{v \in V \text{ s.t.} h_i(v) \geq d_i \log n/96k} h_i(v) \geq \frac{1}{3}\mathrm{Vol}(A_i)$.
- **iii)** None of the above conditions are met, i.e., it holds that $\sum_{v \in S_t \text{ s.t.} h_i(v) \leq d_i \log n/k} h_i(v) < \frac{1}{2}\mathrm{Vol}(A_i)$ and $\sum_{v \in V \text{ s.t.} h_i(v) \geq d_i \log n/96k} h_i(v) < \frac{1}{3}\mathrm{Vol}(A_i)$.

In the remainder of this subsection we prove that in case i) ∂I_t grows, while in cases ii) and iii) it is I_t to grow directly.

Case i)

It holds that

$$\sum_{\substack{v \in S_t \text{ s.t.} \\ h_i(v) \leq d_i \log n/k}} h_i(v) \geq \frac{1}{2}\mathrm{Vol}(A_i). \tag{2}$$

In this case we give a lower bound on the number of new nodes in the boundary and prove that it holds with high probability. We start by noting that the probability that a fixed node $u \in A_i$ pulls the rumor from I_t in a given round is

$$1 - \left(1 - \frac{|N(u) \cap I_t|}{\deg(u)}\right)^k \geq 1 - \left(1 - \frac{1}{\deg(u)}\right)^k \geq 1 - \left(1 - \frac{1}{2d_i}\right)^k \geq \frac{k}{4d_i} =: p,$$

where we use that $2d_i > k$.

Let us pessimistically assume that such a probability exactly equals p for every node u. For each $u \in A_i$, let X_u denote the 0/1 random variable that is 1 iff u pulls the rumor from I_t in round $t+1$. Then we have $\mathbb{P}[X_u = 1] = p$. Further, for each $v \in S_t$ such that $h_i(v) \leq d_i \log n/k$, let Y_v denote the 0/1 random variable which is 1 exactly when v has a neighbor u with $X_u = 1$. Finally let $Y = \sum_{v \in V} Y_v$ be the number of new nodes in the boundary only considering the contribution from pull. Our goal is to prove a lower bound on Y.

We see that

$$\mathbb{P}[Y_v = 1] \geq 1 - \left(1 - \frac{k}{4d_i}\right)^{h_i(v)} \geq 1 - \left(1 - \frac{k}{4d_i}\right)^{h_i(v)/\log n} \geq \frac{kh_i(v)}{8d_i \log n},$$

where the last inequality follows from Taylor series expansion, which needs that $\frac{kh_i(v)}{8d_i \log n} \leq 1$, as ensured by the second to last inequality[4]. This gives us that in expectation

$$\begin{aligned}\mathbb{E}[Y] = \sum_{\substack{v \in S_t \text{ s.t.} \\ h_i(v) \leq d_i \log n/k}} \mathbb{P}[Y_v = 1] &\geq \sum_{\substack{v \in S_t \text{ s.t.} \\ h_i(v) \leq d_i \log n/k}} \frac{kh_i(v)}{8d_i \log n} \\ &\geq \frac{k}{16d_i \log n}\mathrm{Vol}(A_i) \geq \frac{k}{16d_i \log n} \cdot d_i|A_i| = \frac{k}{16 \log n}|A_i|.\end{aligned}$$

[4] Note that without the extra $\log n$ the rightmost term could be bigger than 1, rendering the inequality trivially false.

Next, we need to show that Y is concentrated around its expectation. A simple Chernoff bound does not suffice: the Y_v are not independent, nor negatively correlated. Instead, we use the method of bounded differences (see Theorem 13). In particular, we apply this theorem with $R_i = X_u$, $f(\{X_u\}_{u\in A_i}) = Y$, so $\mu = \mathbb{E}[Y]$ and $b := \max |f(x) - f(x')| \le 2d_i$. With $\lambda = \frac{k}{32\log n}|A_i|$ we get

$$\mathbb{P}\left[Y < \frac{k}{32\log n}|A_i|\right] = \mathbb{P}\left[Y < \frac{k}{16\log n}|A_i| - \frac{k}{32\log n}|A_i|\right]$$
$$\le \exp\left(-\frac{k|A_i|}{32^2(2+1/24)\cdot d_i\log^2 n}\right) \le \exp\left(-\frac{k|A_i|}{2091\cdot d_i\log^2 n}\right).$$

Using that $d_i \le 16|A_i|$, and that $k > \log^3 n$, we get with high probability that

$$|\partial I_{t+1} \setminus \partial I_t| \ge \frac{k}{32\log n}|A_i|.$$

Case ii)

It holds that

$$\sum_{\substack{v\in V \text{ s.t.}\\ h_i(v)\ge d_i\log n/96k}} h_i(v) \ge \frac{1}{3}\operatorname{Vol}(A_i). \tag{3}$$

Intuitively, we want to formalize the fact that the number of informed nodes in A_i must grow even if the nodes are not likely to pull directly from I_t. We do this by looking at a subset A_i' of the nodes in A_i that have many connections. More formally we have the following fact, for a proof we refer to [34].

▷ Claim 10 (Section 3.4 in [34]). Let $\ell = d_i\log n/96k$. Then there exist sets $A_i' \subseteq A_i$ and $V' \subseteq V$ such that each node in A_i' has at least $d_i/8$ neighbors in V' and each node in V' has at least $\ell/8$ neighbors in A_i', and the size of A_i' is at least $|A_i'| \ge |A_i|/20$.

We will show that each node of $s \in A_i'$ gets informed in $O(\log_k n)$ rounds with high probability. By Lemma 15, this is the same as a rumor from s reaching I_t after $O(\log_k n)$ rounds. First, we show that a rumor started at $s \in B$ reaches $\ell/1024$ nodes in A_i' in $O(\log_k n)$ rounds w.h.p. Note that $\ell/1024 \ge 1$ since $d_i \ge 1024\cdot 96k$. Then the probability that one of these nodes pushes the rumor to I_t in the next $c\cdot 2048\cdot 96$ rounds is at least

$$1 - \left(1 - \frac{1}{2d_i}\right)^{k\frac{\ell}{1024}c\cdot 2048\cdot 96} = 1 - \left(1 - \frac{1}{2d_i}\right)^{c\cdot 2d_i\log n} \ge 1 - n^{-c},$$

where the equality follows from the definition of ℓ.

To show that a rumor from $s \in A_i'$ reaches $\ell/1024$ nodes in A_i' in $O(\log_k n)$ rounds, we show that in $\log_k n$ phases of $O(1)$ rounds, the number of informed nodes in A_i' grows by a factor k. We do this in two steps. First we show that by push A_i' informs nodes in V', and then by pull nodes from A_i' pull the rumor from V'.

Let m denote the number of informed nodes of A_i' at the start of the phase. We show that by $O(1)$ push rounds we have at least $\min\{km, \ell/16\}$ informed nodes in V'. Suppose we have less than $\min\{km, \ell/16\}$ informed nodes V_i' in V' after i pushes. Then the probability of a successful push from $u \in A_i'$ is at least

$$\frac{|(N(u)\cap V')\setminus V_i'|}{\deg(u)} \ge \frac{|N(u)\cap V'| - |V_i'|}{\deg(u)} \ge \frac{d_i/8 - \min\{km,\ell/16\}}{2d_i}$$
$$\ge \frac{d_i/8 - d_i/16}{2d_i} \ge \frac{1}{32}.$$

So the expected number of informed nodes in V' after 1 round is at least $\min\left\{\frac{mk}{32}, \ell/16\right\}$. Since these pushes are negatively correlated, Chernoff, Theorem 12, gives that we have with high probability that the number of informed nodes in V' after 1 round is at least

$$\min\left\{\frac{mk}{64}, \ell/16\right\}.$$

Now we consider pulling from V'. Suppose m' nodes V'_{inf} in V' are informed, now the expected number of informed nodes in A'_i after 96 pull rounds is at least

$$\begin{aligned}
&\sum_{u\in A'_i} 1 - \left(1 - \frac{|N(u)\cap V'_{\text{inf}}|}{\deg(u)}\right)^{96k} = \sum_{u\in A'_i} 1 - \left(1 - \frac{|N(u)\cap V'_{\text{inf}}|}{\deg(u)}\right)^{d_i \log n/\ell} \\
&\geq \sum_{u\in A'_i} 1 - \left(1 - \frac{|N(u)\cap V'_{\text{inf}}|}{2d_i}\right)^{\frac{d_i}{\ell}} \geq \sum_{u\in A'_i} \frac{d_i}{\ell}\frac{|N(u)\cap V'_{\text{inf}}|}{4d_i} \\
&= \frac{1}{4\ell}\sum_{v\in V'_{\text{inf}}} |N(v)\cap A'_i| \geq \frac{1}{4\ell}\sum_{v\in V'_{\text{inf}}} \frac{\ell}{8} = \frac{|V'_{\text{inf}}|}{32},
\end{aligned}$$

where the second inequality uses that $|N(u)\cap V'_{\text{inf}}| \leq |V'_{\text{inf}}| \leq \ell/16$. So we conclude that in expectation, we have at least $\min\left\{\frac{mk}{32\cdot 64}, \frac{\ell}{32\cdot 16}\right\}$ informed nodes in A'_i. Since all pulls are independent, Chernoff, Theorem 12, gives us that with high probability we have at least

$$\min\left\{\frac{mk}{4096}, \frac{\ell}{1024}\right\}.$$

informed nodes in A'_i.

We conclude that in $r = O(\log_k n)$ rounds we have

$$|I_{t+r} \setminus I_t| \geq |A'_i| \geq \frac{|A_i|}{20},$$

where the last inequality comes from Claim 10.

Case iii)

None of the previous conditions holds, i.e., we have that

$$\sum_{\substack{v\in S_t \text{ s.t.}\\ h_i(v)\leq d_i \log n/k}} h_i(v) < \frac{1}{2}\operatorname{Vol}(A_i) \qquad \text{and} \qquad \sum_{\substack{v\in V \text{ s.t.}\\ h_i(v)\geq d_i \log n/96k}} h_i(v) < \frac{1}{3}\operatorname{Vol}(A_i).$$

By using the two above conditions on the volume of A_i, we can see that the number of edges going from A_i to some $v \in V \setminus S_t$ with $h_i(v) < d_i/96k$ is at least a constant fraction of the volume of A_i, namely

$$\begin{aligned}
\sum_{\substack{v\in V\setminus S_t \text{ s.t.}\\ h_i(v)< d_i \log n/96k}} h_i(v) &= \sum_{v\in V} h_i(v) - \sum_{\substack{v\in V \text{ s.t.}\\ h_i(v)\geq d_i \log n/96k}} h_i(v) - \sum_{\substack{v\in S_t \text{ s.t.}\\ h_i(v)< d_i \log n/96k}} h_i(v) \\
&> \operatorname{Vol}(A_i) - \sum_{\substack{v\in V \text{ s.t.}\\ h_i(v)\geq d_i \log n/96k}} h_i(v) - \sum_{\substack{v\in S_t \text{ s.t.}\\ h_i(v)\leq d_i \log n/k}} h_i(v) \\
&\geq \frac{1}{6}\operatorname{Vol}(A_i).
\end{aligned}$$

Moreover, the number of such edges going to ∂I_t is at most $\frac{d_i \log n}{96k}|\partial I_t|$, since $h_i(v) < d_i \log n/96k$. Therefore, using the condition on $|A_i|$ in Equation (1), that $\deg(u) \geq d_i$ for every $u \in A_i$, and that $k > \log^2 n$, we get

$$|E(A_i, I_t)| \geq \frac{1}{6}\operatorname{Vol}(A_i) - \frac{d_i \log n}{96k}|\partial I_t| \geq \frac{d_i|A_i|}{6} - \frac{d_i \log n}{96k}(4 \log n \cdot |A_i|) \geq \frac{d_i|A_i|}{8}. \tag{4}$$

Let $B \subseteq A_i$ be the set of nodes in A_i that have at least $d_i/16$ neighbors in I_t. Then, using that $\deg(u) < 2d_i$ for every $u \in A_i$, we get

$$|E(A_i, I_t)| \leq |B| \cdot 2d_i + (|A_i| - |B|)\frac{d_i}{16}. \tag{5}$$

By combining Equations (4) and (5) it follows that $|B| \geq \frac{|A_i|}{32}$.

Note that each node $u \in B$ in the set has a fraction of at least $\frac{d_i/16}{\deg(u)} \geq \frac{d_i/16}{2d_i} = 1/32$ neighbors in I_t. We can then apply Lemma 9 and get that with high probability at least $|B|/4 \geq |A_i|/128$ nodes in A_i pull the rumor from I_t in at most $\lceil 1/\lceil k/32 \rceil \rceil = 1$ round, since $k > \log^2 n$. Hence with high probability it follows that

$$|I_{t+1} \setminus I_t| \geq \frac{|A_i|}{128}.$$

3.3 Sets with high degree nodes

Recall that in this case every $u \in A_i$ is such that

$$d_i \geq 2|\partial I_t|.$$

As in the previous case, also here nodes can either contribute directly to the growth of I_t or to that of the boundary ∂I_t. Recall that $S_t := V \setminus (I_t \cup \partial I_t)$. Let us denote the set of nodes in A_i that have more neighbors toward S_t than I_t as

$$B := \{u \in A_i : |N(u) \cap S_t| \geq k|N(u) \cap I_t|\}.$$

We distinguish three cases:

- **i)** $|B| \geq \frac{1}{2}|A_i|$, and $\deg(u) \geq \sqrt{k}|B|$.
- **ii)** $|B| \geq \frac{1}{2}|A_i|$, $\deg(u) < \sqrt{k}|B|$.
- **iii)** $|B| < \frac{1}{2}|A_i|$.

As before in the remainder of this subsection we prove separately for each of these cases that either I_t or ∂I_t grows sufficiently. In particular in cases i) and ii) the boundary grows sufficiently, while in case iii) it is I_t to grow.

Case i)

It holds that

$$|B| \geq \frac{1}{2}|A_i| \quad \text{and} \quad \deg(u) \geq \sqrt{k}|B|.$$

In this case, we start by showing that after $\lceil \frac{|I_t|+|\partial I_t|}{|B| \log n} \rceil$ rounds, we inform at least one node $u \in B$ with high probability.

The probability that a fixed node $v \in I_t$ does not push to any neighbor in B is $1 - \frac{h_i(v)}{\deg(v)} \leq 1 - \frac{h_i(v)}{|I_t|+|\partial I_t|}$. Hence, the probability that no $v \in I_t$ pushes to B within $\frac{|I_t|+|\partial I_t|}{|B|\log n}$ rounds of k-PUSH&PULL is upper bounded by

$$\prod_{v \in I_t} \left(1 - \frac{h_i(v)}{|I_t| + |\partial I_t|}\right)^{k \frac{|I_t|+|\partial I_t|}{|B|\log n}} \leq e^{-k/\log n \frac{1}{|B|} \sum_{v \in I_t} h_i(v)} \leq e^{-k/\log n}.$$

This uses that $\sum_{v \in I_t} h_i(v) = \sum_{v \in A_i} |N(u) \cap I_t| \geq |A_i| \geq |B|$, since $A_i \subseteq \partial I_t$.

Hence, the probability that push informs at least one node in these rounds is at least $1 - e^{-k/\log n}$, namely with high probability, since $k > \log^2 n$.

Therefore, since $\deg(u) \geq \sqrt{k}|B|$, and by definition of the set B, we get with high probability that

$$|\partial I_{t+r} \setminus \partial I_t| \geq \frac{\sqrt{k}|B|}{2} \geq \frac{\sqrt{k}|A_i|}{4} \geq \frac{\sqrt{k}|\partial I_t|}{16 \log n}$$

for a number of rounds $r = \lceil \frac{|I_t|+|\partial I_t|}{|B|\log n} \rceil = O(1)$ since $|B| \geq \frac{1}{2}|A_i| \geq \frac{|\partial I_t|}{8 \log n}$, where $|A_i| \geq \frac{|\partial I_t|}{4 \log n}$ since we consider $i \in \mathcal{I}$.

Case ii)

It holds that

$$|B| \geq \frac{1}{2}|A_i| \quad \text{and} \quad \deg(u) < \sqrt{k}|B|.$$

We start by noting that the probability that a fixed node $u \in A_i$ pulls the rumor from I_t in a given round is

$$1 - \left(1 - \frac{|N(u) \cap I_t|}{\deg(u)}\right)^k \geq 1 - \left(1 - \frac{1}{\deg(u)}\right)^k \geq 1 - \left(1 - \frac{1}{2d_i}\right)^k \geq \frac{k}{4d_i} =: p,$$

where we use in the last inequality that $2d_i > k$.

Formally, let X_u be the 0/1 random variable that is 1 iff u pulls the rumor. Note that the X_u's are independent since they come from pull operations. Let us pessimistically assume that such a probability exactly equals p for every node u. For each node $v \in N(B) \cap S_t$ let Y_v be the 0/1 random variable that is 1 iff v has at least a neighbor u with $X_u = 1$.

We look at $v \in N(B) \cap S_t$ and consider the probability that no neighbor of v in B pulls the rumor, which is at least

$$1 - \prod_{u \in B \cap N(v)} \left(1 - \frac{k}{4d_i}\right) \geq 1 - \exp\left(-k/4 \sum_{u \in B \cap N(v)} \frac{1}{d_i}\right) \geq \frac{k}{8} \sum_{u \in B \cap N(v)} \frac{1}{d_i},$$

where the last inequality follows from the assumption on nodes $u \in B$. The expected number of nodes $v \in N(B) \cap S_t$ that join ∂I_t is now at least

$$\sum_{v \in N(B) \cap S_t} \frac{k}{8} \sum_{u \in B \cap N(v)} \frac{1}{d_i} = \frac{k}{8} \sum_{u \in B} \frac{|N(u) \cap S_t|}{d_i} \geq \frac{k}{16}|B|,$$

where the second to last equality follows by definition of B and the fact that $d_i > 2|\partial I_t|$.

We now show that the expected number of new nodes in the boundary is not far from that lower bound. Formally we lower bound $Y := \sum_{v \in N(B) \cap S_t} Y_v$. Hereto, we use the method of bounded differences (Theorem 13) with $R_i = X_u$, $f(\{X_u\}_{u \in B}) = Y$, and $b \leq 2d_i$. Since $\mathbb{E}[Y] \geq \frac{k}{16}|B|$, with $\lambda = \frac{k}{32}|B|$ we get

$$\mathbb{P}\left(Y < \frac{k}{32}|B|\right) \leq \exp\left(-\frac{k|B|}{2091 \cdot d_i}\right).$$

Since $\deg(u) < 2d_i < 2\sqrt{k}|B|$ and $k > \log^2 n$, the previous bound holds with high probability. Therefore with high probability we have

$$|\partial I_{t+1} \setminus \partial I_t| \geq \frac{k}{64}|A_i|.$$

Case iii)

It holds that

$$|B| < \frac{1}{2}|A_i|.$$

It follows directly from the above condition that $|A_i \setminus B| > |A_i|/2$ and by the definition of B that each node $u \in A_i \setminus B$ iff $|N(u) \cap S_t| < k|N(u) \cap I_t|$. Recall that we are in the case of high degree nodes: $|N(u)| \geq 2|\partial I_t|$, equivalently, we have $|\partial I_t| \leq \frac{1}{2}|N(u)|$. Hence $|N(u) \cap \partial I_t| \leq |\partial I_t| \leq \frac{1}{2}|N(u)|$, and $|N(u) \cap I_t| + |N(u) \cap S_t| \geq \frac{1}{2}|N(u)|$, so $|N(u) \cap \partial I_t| \leq |N(u) \cap I_t| + |N(u) \cap S_t|$. Combining this, we obtain

$$\begin{aligned}\frac{|N(u) \cap I_t|}{|N(u)|} &= \frac{|N(u) \cap I_t|}{|N(u) \cap I_t| + |N(u) \cap S_t| + |N(u) \cap \partial I_t|} \\ &\geq \frac{|N(u) \cap I_t|}{2(|N(u) \cap I_t| + |N(u) \cap S_t|)} > \frac{|N(u) \cap I_t|}{2(k+1)|N(u) \cap I_t|} = \frac{1}{2(k+1)}.\end{aligned}$$

We conclude that u has a $\frac{1}{2(k+1)}$-fraction of all its neighbors in I_t. Then we apply Lemma 9 and get that with high probability at least one fourth of the nodes in $|A_i \setminus B|$ pull the rumor from I_t in at most $\lceil \frac{2(k+1)}{k} \rceil = O(1)$ rounds. Hence we get with high probability that

$$|I_{t+r} \setminus I_t| \geq \frac{|A_i \setminus B|}{4} \geq \frac{|A_i|}{8}$$

for a number of rounds $r = O(1)$.

References

1. Hüseyin Acan, Andrea Collevecchio, Abbas Mehrabian, and Nick Wormald. On the push&pull protocol for rumour spreading: [extended abstract]. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC 2015)*, pages 405–412. ACM, 2015. doi:10.1145/2767386.2767416.
2. Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. *Combinatorics, Probability & Computing*, 20(4):481–502, 2011. Announced at SPAA'08. doi:10.1017/S0963548311000125.
3. Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM*, 62(5):42:1–42:25, 2015. Announced at FOCS 2010. doi:10.1145/2775105.
4. Sanjeev Arora, Frank Thomson Leighton, and Bruce M. Maggs. On-line algorithms for path selection in a nonblocking network. *SIAM J. Comput.*, 25(3):600–625, 1996. Announced at STOC '90. doi:10.1137/S0097539791221499.

5 Richard Arratia and Louis Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of mathematical biology*, 51(1):125–131, 1989.

6 Chen Avin and Robert Elsässer. Breaking the $\log n$ barrier on rumor spreading. *Distributed Computing*, 31(6):503–513, 2018. Announced at DISC 2013. doi:10.1007/S00446-017-0312-4.

7 Petra Berenbrink, Robert Elsässer, and Tom Friedetzky. Efficient randomised broadcasting in random regular networks with applications in peer-to-peer systems. *Distributed Comput.*, 29(5):317–339, 2016. Announced at PODC 2008. doi:10.1007/S00446-016-0264-0.

8 Keren Censor-Hillel, Bernhard Haeupler, Jonathan A. Kelner, and Petar Maymounkov. Rumor spreading with no dependence on conductance. *SIAM Journal on Computing*, 46(1):58–79, 2017. Announced at STOC 2012. doi:10.1137/14099992X.

9 Keren Censor-Hillel and Hadas Shachnai. Partial information spreading with application to distributed maximum coverage. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC 2010)*, pages 161–170. ACM, 2010. doi:10.1145/1835698.1835739.

10 Keren Censor-Hillel and Hadas Shachnai. Fast information spreading in graphs with large weak conductance. *SIAM Journal on Computing*, 41(6):1451–1465, 2012. Announced at SODA 2011. doi:10.1137/110845380.

11 Flavio Chierichetti, George Giakkoupis, Silvio Lattanzi, and Alessandro Panconesi. Rumor spreading and conductance. *Journal of the ACM*, 65(4):17:1–17:21, 2018. doi:10.1145/3173043.

12 Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Almost tight bounds for rumour spreading with conductance. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 399–408. ACM, 2010. doi:10.1145/1806689.1806745.

13 Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Rumor spreading in social networks. *Theoretical Computer Science*, 412(24):2602–2610, 2011. Announced at ICALP 2009. doi:10.1016/J.TCS.2010.11.001.

14 Eden Chlamtác, Michael Dinitz, and Yury Makarychev. Minimizing the union: Tight approximations for small set bipartite vertex expansion. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 881–899. SIAM, 2017. doi:10.1137/1.9781611974782.56.

15 Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, 2001. doi:10.1006/AAMA.2001.0720.

16 Julia Chuzhoy, Yury Makarychev, Aravindan Vijayaraghavan, and Yuan Zhou. Approximation algorithms and hardness of the k-route cut problem. *ACM Transactions on Algorithms*, 12(1):2:1–2:40, 2016. Announced at SODA 2012. doi:10.1145/2644814.

17 Sebastian Daum, Fabian Kuhn, and Yannic Maus. Rumor spreading with bounded in-degree. *Theoretical Computer Science*, 810:43–57, 2020. Announced at SIROCCO 2016. doi:10.1016/J.TCS.2018.05.041.

18 Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. In *Learning on Graphs Conference*, pages 38–1. PMLR, 2022.

19 Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. *ACM SIGOPS Oper. Syst. Rev.*, 22(1):8–32, 1988. Announced at PODC 1987. doi:10.1145/43921.43922.

20 Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. *Electronic Notes in Discrete Mathematics*, 38:297–302, 2011. Announced at ICALP 2011. doi:10.1016/J.ENDM.2011.09.049.

21 Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. *Electronic Notes in Discrete Mathematics*, 38:303–308, 2011. Announced at STOC 2011. doi:10.1016/J.ENDM.2011.09.050.

22 Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Asynchronous rumor spreading in preferential attachment graphs. In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2012)*, volume 7357 of *LNCS*, pages 307–315. Springer, 2012. doi:10.1007/978-3-642-31155-0_27.

23 Benjamin Doerr, Tobias Friedrich, and Thomas Sauerwald. Quasirandom rumor spreading. *ACM Transactions on Algorithms*, 11(2):9:1–9:35, 2014. Announced at SODA 2008. doi:10.1145/2650185.

24 Benjamin Doerr and Anatolii Kostrygin. Randomized rumor spreading revisited. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *LIPIcs*, pages 138:1–138:14. LZI, 2017. doi:10.4230/LIPICS.ICALP.2017.138.

25 Benjamin Doerr and Marvin Künnemann. Tight analysis of randomized rumor spreading in complete graphs. In *Proceedings of the Eleventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO 2014)*, pages 82–91. SIAM, 2014. doi:10.1137/1.9781611973204.8.

26 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

27 Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures & Algorithms*, 1(4):447–460, 1990. Announced at SIGAL 1990. doi:10.1002/RSA.3240010406.

28 Nikolaos Fountoulakis and Konstantinos Panagiotou. Rumor spreading on random regular graphs and expanders. *Random Structures & Algorithms*, 43(2):201–220, 2013. Announced at APPROX-RANDOM 2010. doi:10.1002/RSA.20432.

29 Nikolaos Fountoulakis, Konstantinos Panagiotou, and Thomas Sauerwald. Ultra-fast rumor spreading in social networks. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1642–1660. SIAM, 2012. doi:10.1137/1.9781611973099.130.

30 Alan M. Frieze and Geoffrey R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985. doi:10.1016/0166-218X(85)90059-9.

31 Mohsen Ghaffari and Calvin C. Newport. How to discreetly spread a rumor in a crowd. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC 2016)*, volume 9888 of *LNCS*, pages 357–370. Springer, 2016. doi:10.1007/978-3-662-53426-7_26.

32 George Giakkoupis. Tight bounds for rumor spreading with vertex expansion. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 801–815. SIAM, 2014. doi:10.1137/1.9781611973402.59.

33 George Giakkoupis, Yasamin Nazari, and Philipp Woelfel. How asynchrony affects rumor spreading time. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC 2016)*, pages 185–194. ACM, 2016. doi:10.1145/2933057.2933117.

34 George Giakkoupis and Thomas Sauerwald. Rumor spreading and vertex expansion. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1623–1641. SIAM, 2012. doi:10.1137/1.9781611973099.129.

35 Bernhard Haeupler. Simple, fast and deterministic gossip and rumor spreading. *Journal of the ACM*, 62(6):47:1–47:18, 2015. Announced at SODA 2013. doi:10.1145/2767126.

36 Bernhard Haeupler and Dahlia Malkhi. Optimal gossip with direct addressing. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 176–185, 2014. doi:10.1145/2611462.2611489.

37 Jun-Ting Hsieh, Alexander Lubotzky, Sidhanth Mohanty, Assaf Reiner, and Rachel Yun Zhang. Explicit lossless vertex expanders. *arXiv preprint arXiv:2504.15087*, 2025. doi:10.48550/arXiv.2504.15087.

38 Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer*

Science (FOCS 2000), pages 565–574. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892324.

39 Anand Louis and Yury Makarychev. Approximation algorithms for hypergraph small-set expansion and small-set vertex expansion. *Theory of Computing*, 12(1):1–25, 2016. Announced at APPROX-RANDOM 2014. doi:10.4086/TOC.2016.V012A017.

40 Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, pages 195–248. Springer, 1998.

41 Abbas Mehrabian and Ali Pourmiri. Randomized rumor spreading in poorly connected small-world networks. *Random Structures & Algorithms*, 49(1):185–208, 2016. Announced at DISC 2014. doi:10.1002/RSA.20624.

42 Yves Mocquard, Bruno Sericola, and Emmanuelle Anceaume. Analysis of rumor spreading with 2-pull or 3-pull operations. In *Proceedings of the 20th IEEE International Symposium on Network Computing and Applications (NCA 2021)*, pages 1–8. IEEE, 2021. doi:10.1109/NCA53618.2021.9685439.

43 Guy Moshkovitz and Asaf Shapira. Decomposing a graph into expanding subgraphs. *Random Structures & Algorithms*, 52(1):158–178, 2018. Announced at SODA 2015. doi:10.1002/RSA.20727.

44 Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008. Announced at PODC 2006. doi:10.1109/TIT.2008.924648.

45 Charlotte Out, Nicolás Rivera, Thomas Sauerwald, and John Sylvester. Rumors with changing credibility. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 86:1–86:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.86.

46 Konstantinos Panagiotou, Xavier Pérez-Giménez, Thomas Sauerwald, and He Sun. Randomized rumour spreading: The effect of the network topology. *Combinatorics, Probability & Computing*, 24(2):457–479, 2015. doi:10.1017/S0963548314000194.

47 Konstantinos Panagiotou, Ali Pourmiri, and Thomas Sauerwald. Faster rumor spreading with multiple calls. *The Electronic Journal of Combinatorics*, 22(1):1, 2015. Announced at ISAAC 2013. doi:10.37236/4314.

48 Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987. doi:10.1137/0147013.

49 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 755–764. ACM, 2010. doi:10.1145/1806689.1806792.

50 Frédérique Robin, Bruno Sericola, Emmanuelle Anceaume, and Yves Mocquard. Stochastic analysis of rumor spreading with multiple pull operations. *Methodology and Computing in Applied Probability*, 24(3):2195–2211, 2022.

51 Naser T Sardari. Diameter of ramanujan graphs and random cayley graphs. *Combinatorica*, 39(2):427–446, 2019. doi:10.1007/S00493-017-3605-0.

52 Thomas Sauerwald and Alexandre Stauffer. Rumor spreading and vertex expansion on regular graphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 462–475. SIAM, 2011. doi:10.1137/1.9781611973082.37.

53 Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *Proceedings of the International Conference on Machine Learning (ICML 2023)*, volume 202 of *Proceedings of Machine Learning Research*, pages 31613–31632. PMLR, 2023. arXiv:2303.06147.

54 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 81–90. ACM, 2004. doi:10.1145/1007352.1007372.

A Concentration Bounds

▶ **Definition 11** (Negatively Associated Random Variables). *The random variables $X_1, \dots, X_n$ are said to be negatively associated if for all disjoint subsets $I, J \subseteq \{1, \dots, n\}$ and all non-decreasing functions f and g it holds that*

$$\mathbb{E}[f(X_i, i \in I) \cdot g(X_j, j \in J)] \leq \mathbb{E}[f(X_i, i \in I)] \cdot \mathbb{E}[g(X_j, j \in J)].$$

▶ **Theorem 12** (Chernoff Bounds [26]). *Let $X_1, \dots, X_n$ be independent $0/1$ random variables. Let $X := \sum_{i=1}^{n} X_i$ and let $\mu_L \leq \mathbb{E}[X] \leq \mu_H$. It holds that*

$$\mathbb{P}[X < (1-\delta)\mu_L] \leq e^{-\frac{\delta^2}{2}\mu_L}, \quad \forall \delta \in (0,1);$$

$$\mathbb{P}[X > (1+\delta)\mu_H] \leq e^{-\frac{\delta^2}{2+\delta}\mu_H}, \quad \forall \delta > 0.$$

The bounds also hold if $X_1, \dots, X_n$ are negatively associated.

▶ **Theorem 13** (Bounded Differences Inequality [40]). *Let $R_1, \dots R_n$ be independent $0/1$ random variables with $\mathbb{P}[R_i = 1] \leq p \leq 1/2$. Let f be a bounded real function defined on $\{0,1\}^n$. Define $\mu := \mathbb{E}[f(R_1, \dots, R_n)]$, and $b := \max |f(x) - f(x')|$, where the maximum is over all $x, x' \in \{0,1\}^n$ that differ only in one position. Then for any $\lambda > 0$*

$$\mathbb{P}[f(R_1, \dots, R_n) \leq \mu - \lambda] \leq \exp\left(-\frac{\lambda^2}{2pnb^2 + 2b\lambda/3}\right).$$

▶ **Theorem 14** ([5]). *Let $B(n,p)$ denote a random variable following the binomial distribution with n trials and success probability p. Then, for $p < a < 1$, we have that*

$$\mathbb{P}[B(n,p) \geq an] \leq \exp\left(-n\left(a \log\left(\frac{a}{p}\right) + (1-a)\log\left(\frac{1-a}{1-p}\right)\right)\right).$$

B Warm-Up: Multi-Call Rumor Spreading on Complete Graphs

This section serves as a warm-up for our main result; we investigate what impact parallel calls have on rumor spreading on complete graphs. We provide a full characterization with matching upper and lower bounds. For $k = 1$, we know that rumor spreading takes $\Theta(\log n)$ rounds [30, 38], so we only consider $k \geq 2$. The analysis in [24] yields precise results that include the leading constant (in fact tight apart from an additive number of rounds) for the case $k = O(1)$. However, it does not extend to super-constant values of k, where it leads to an upper bound of $O(k \cdot \log_k n)$, which is asymptotically worse than ours.

The proofs in the section are generalizations of standard arguments for rumor spreading on cliques. Independent work by Out, Rivera, Sauerwald, and Sylvester [45] considers rumor spreading with a time-dependent credibility function. Some of their proofs resemble the results in this section.

In the proof for the upper bound, we use the following standard lemma, which is a trivial generalization of the version with $k = 1$ (see, e.g., [12, Lemma 3], [34, Lemma 3.3], or [8, Lemma 2.1]).

▶ **Lemma 15.** *For $S, T \subseteq V$, let $T_{k\text{-PUSH\&PULL}}(S,T)$ be the number of rounds for k-PUSH&PULL until a rumor that is initially known to all nodes in S to spread to at least one node of $v \in T$. Let $V_1, V_2 \subseteq V$. Then the random variables $T_{k\text{-PUSH\&PULL}}(V_1, V_2)$ and $T_{k\text{-PUSH\&PULL}}(V_2, V_1)$ have the same distribution.*

▶ **Lemma 16.** *Rumor spreading in a complete graph with the k-PUSH&PULL protocol for every $k \geq 2$ requires $O(\log_k n)$ rounds with high probability.*

Proof. An upper bound of $O(\log n)$ has been known for a long time for $k = 1$ [30, 38]. Since parallel spreading can only make the process faster, we have this upper-bound for any k. In particular, for any constant k we have that the k-PUSH&PULL protocol requires $O(\log n) = O(\log_k n)$ rounds with high probability. In the following we assume $k \geq 17$.

Let I_t denote the informed nodes at round t. First we show using only k-PUSH that $|I_t| \geq (k/4)^t$ until $k|I_t| > n-1$. Assume $k|I_t| \leq n-1$. Each node receives a message from at least one node in I_t with probability $1-(1-\frac{1}{n-1})^{k|I_t|} = \frac{k|I_t|}{n-1} - \frac{1}{2}k|I_t|(k|I_t|-1)\frac{1}{(n-1)^2} + \cdots \geq \frac{k|I_t|}{2(n-1)}$ where we use binomial expansion. Now we see that

$$\mathbb{E}[|I_{t+1} \setminus I_t|] \geq (n-|I_t|)\frac{k|I_t|}{2(n-1)} \geq \left(n - \frac{n-1}{k}\right)\frac{k|I_t|}{2(n-1)} \geq \tfrac{k-1}{2}|I_t|.$$

$|I_{t+1} \setminus I_t|$ can be written as a sum of 0/1 random variables. Note that they are not independent, but they are negatively associated since the probability that u receives a push can only decrease under the assumption that v receives a push. Hence, as discussed in the appendix (see Theorem 12), Chernoff still gives

$$\mathbb{P}[|I_{t+1} \setminus I_t| < \tfrac{k-1}{4}|I_t|] \leq e^{-(k-1)|I_t|/16}.$$

For $k \geq 16(c+1)\log n$, this is bounded by $n^{-(c+1)}$, which gives the required probability. For smaller k, we need to analyze the process more carefully.

We consider $\beta \cdot 2\log_k n$ rounds, for some constant $\beta = \beta(c) \geq 2$ to be decided later. By a Chernoff bound, the probability that any such round does *not* have the required growth is bounded by $e^{-(k-1)/16}$. We compute the probability that more than $(\beta-1)\cdot 2\log_k n$ rounds fail in reaching the required growth. In fact, if at most $(\beta-1)\cdot 2\log_k n$ rounds fail, we have at least $2\log_k n \geq \frac{\log(n/k)}{\log(k/4)}$ successes which is enough to reach $|I_t| > (n-1)/k$ nodes.

To count the number of failures, we see that this is dominated by a binomial random variable $B(N,p)$ with $N = \beta \cdot 2\log_k n$ trials and probability of success $p = e^{-(k-1)/16}$. We bound the probability that $B(N,p) \geq (\beta-1)\cdot 2\log_k n = \frac{\beta-1}{\beta}N$. By using standard tail bounds on the binomial distribution (see Theorem 14) we get that for $p < \frac{\beta-1}{\beta}$ we have

$$\mathbb{P}\left[B(N,p) \geq \frac{\beta-1}{\beta}N\right] \leq \exp\left(-N\left(\frac{\beta-1}{\beta}\log\left(\frac{\frac{\beta-1}{\beta}}{p}\right) + \left(1-\frac{\beta-1}{\beta}\right)\log\left(\frac{1-\frac{\beta-1}{\beta}}{1-p}\right)\right)\right).$$

Note that we have $p < \frac{\beta-1}{\beta}$, since $p = e^{-(k-1)/16} \leq e^{-1} < 1/2 \leq \frac{\beta-1}{\beta}$, where the last inequality follows from the assumption that $\beta \geq 2$.

Now we simplify this bound

$$\begin{aligned}
\mathbb{P}\left[B(N,p) \geq \frac{\beta-1}{\beta}N\right] &\leq \exp\left(-N\left(\frac{\beta-1}{\beta}\log\left(\frac{\frac{\beta-1}{\beta}}{p}\right) + \left(1-\frac{\beta-1}{\beta}\right)\log\left(\frac{1-\frac{\beta-1}{\beta}}{1-p}\right)\right)\right) \\
&= \exp\left(-\beta 2\log_k(n)\left(\frac{\beta-1}{\beta}\log\left(\frac{\frac{\beta-1}{\beta}}{e^{-(k-1)/16}}\right) + \frac{1}{\beta}\log\left(\frac{\frac{1}{\beta}}{1-e^{-(k-1)/16}}\right)\right)\right) \\
&\leq \exp\left(-2\log_k(n)\left((\beta-1)\left(\log\left(\frac{\beta-1}{\beta}\right) + (k-1)/16\right) - \log\beta\right)\right).
\end{aligned}$$

We add the constraint that $\beta \geq 3$, so that $\log\left(\frac{\beta-1}{\beta}\right) > -1/2 \geq -(k-1)/32$, hence we have

$$\mathbb{P}\left[B(N,p) \geq \frac{\beta-1}{\beta}N\right] \leq \exp\left(-2\log_k(n)\cdot(\beta-1)(k-1)/32 - \log\beta\right)$$
$$= \exp\left(-2\log_k(n)\cdot(\beta-1)\left((k-1)/32 - \frac{\log\beta}{\beta-1}\right)\right).$$

We add the further constraint that $\beta \geq 11$, so that $\frac{\log\beta}{\beta-1} < 1/4 \leq (k-1)/64$. This gives

$$\mathbb{P}\left[B(N,p) \geq \frac{\beta-1}{\beta}N\right] \leq \exp\left(-2\log_k(n)\cdot(\beta-1)(k-1)/64\right)$$
$$= \exp\left(-\log(n)\cdot\frac{\beta-1}{32}\cdot\frac{k-1}{\log k}\right) \leq n^{-(c+1)},$$

where the last inequality holds for $\beta \geq 32c+33$. In other words, we have shown that with high probability after at most $O(\log_k n)$ rounds we have that $k|I_t| > n-1$.

Next, we show that $|I_{t+1}| > n/2$ w.h.p., so assume $|I_t| \leq n/2$. Let $v \in V \setminus I_t$. Then the probability that v pulls from I_t is

$$1 - \left(1 - \tfrac{|I_t|}{n-1}\right)^k \geq 1 - (1-\tfrac{1}{k})^k \geq 1 - 1/e.$$

So the expected number of nodes that pull is at least $(n-|I_t|)(1-1/e)$. In this case, $|I_{t+1}\setminus I_t|$ is a standard binomial random variable (i.e., sum of independent 0/1 random variables). Hence, using Chernoff we see that

$$\mathbb{P}[|I_{t+1}\setminus I_t| \leq \tfrac{n-|I_t|}{2}] = \mathbb{P}[|I_{t+1}\setminus I_t| \leq (1-\tfrac{1+1/e}{2})(n-|I_t|)(1-1/e)]$$
$$\leq e^{-\left(\frac{1+1/e}{2}\right)^2(n-|I_t|)(1-1/e)/2} \leq e^{-n/14},$$

using that $n-|I_t| \geq n/2$. Now we have that $|I_{t+1}| > |I_t| + \frac{n-|I_t|}{2} \geq n/2$.

By a standard symmetry argument, see Lemma 15, we see that we use at most $2O(\log_k n) = O(\log_k n)$ rounds with probability $1-n^{-c}$. ◀

Next, we show the matching lower bound.

▶ **Lemma 17.** *Rumor spreading in a complete graph with the k-PUSH&PULL protocol for every $k \geq 2$ requires $\Omega(\log_k n)$ rounds with high probability.*

Proof. We show that it takes $\Omega(\log_k n)$ rounds to get n/k nodes informed. First, let us assume that we are at a round t_0 such that $54(c+1)\log n \leq |I_{t_0}| \leq 216(c+1)\,k\log n$ for some positive constant c. We will later show that this round exists w.h.p. With such an assumption, we show by induction that $|I_{t_0+t}| \leq (3k)^t|I_{t_0}|$, until $|I_t| \geq n/k$.[5] Clearly k-PUSH can inform at most $k|I_t|$ nodes in one round. So it remains to show that the new informed nodes due to k-PULL in one round are at most $2k|I_t|$ w.h.p.

Let $v \in V \setminus I_t$. Then the probability that v pulls from I_t equals

$$1 - \left(1 - \frac{|I_t|}{n-1}\right)^k \leq \frac{3}{2}k\frac{|I_t|}{n-1}.$$

[5] Note that this requires that $216(c+1)\,k\log n < n/k \iff k < \sqrt{n/(216(c+1)\log n)}$, which we can assume since $1 = \Omega(\log_k n)$ is a trivial lower bound for such k.

So we can bound the expected gain from pull $|I_{t+1}^{(\text{pull})}|$ by

$$\mathbb{E}[|I_{t+1}^{(\text{pull})}|] \leq (n - |I_t|) \cdot \frac{3}{2} k \frac{|I_t|}{n-1} \leq \frac{3}{2} k |I_t|.$$

Note that $|I_{t+1}^{(\text{pull})}|$ is a binomial random variable, hence we can use Chernoff to bound the probability that this is bigger than $2k|I_t|$ and get

$$\mathbb{P}[|I_{t+1}^{(\text{pull})}| > 2k|I_t|] = \mathbb{P}\left[|I_{t+1}^{(\text{pull})}| > \left(1 + \frac{1}{3}\right)\frac{3}{2}k|I_t|\right] \leq e^{-\mathbb{E}[|I_{t+1}^{(\text{pull})}|]/27}.$$

To see that this probability is indeed low, we lower bound the expected increase $|I_{t+1}^{(\text{pull})}|$ as well. The probability that any $v \in V \setminus I_t$ pulls from I_t equals

$$1 - \left(1 - \frac{|I_t|}{n-1}\right)^k \geq k \frac{|I_t|}{2(n-1)},$$

since we are assuming $|I_t| \leq n/k$. So we see

$$\mathbb{E}[|I_{t+1}^{(\text{pull})}|] \geq (n - |I_t|)k\frac{|I_t|}{2(n-1)} \geq \frac{n - n/k}{2(n-1)} k|I_t| = \frac{n(k-1)}{2(n-1)}|I_t| \geq \frac{k-1}{2}|I_t| \geq \frac{|I_t|}{2}.$$

Since we have by assumption that $|I_t| \geq |I_{t_0}| \geq 54(c+1)\log n$ the result holds at each step with probability at least $1 - n^{-(c+1)}$. If $|I_{t_0+t}| \leq (3k)^t |I_{t_0}|$, then we need

$$\frac{\log\left(\frac{n/k}{|I_{t_0}|}\right)}{\log(3k)} = \Omega(\log_k n)$$

rounds to inform at least n/k nodes.

It remains to show that until we inform $54(c+1)\log n$ nodes, we cannot suddenly overshoot $216(c+1)\,k\log n$ nodes. We note that $I_{t+1} = I_t \cup I_{t+1}^{(\text{push})} \cup I_{t+1}^{(\text{pull})}$. It is clear that k-PUSH can inform at most $k \cdot 54(c+1)\log n$ nodes. So it remains to show that it is very unlikely that the process reaches more than $216(c+1)\,k\log n$ nodes through k-PULL. We do this by applying a Chernoff bound in a different way. More precisely, assume that $|I_t| \leq 54(c+1)\log n$, then we show that the probability that $|I_{t+1}^{(\text{pull})}| > 108(c+1)\,k\log n$ is small:

$$\mathbb{P}[|I_{t+1}^{(\text{pull})}| > 108(c+1)\,k\log n] = \mathbb{P}\left[|I_{t+1}^{(\text{pull})}| > \left(1 + \left(\frac{1}{2} + \frac{3}{2}\right)\frac{(54(c+1)\,k\log n}{\mathbb{E}[|I_{t+1}^{(\text{pull})}|]} - 1\right)\mathbb{E}[|I_{t+1}^{(\text{pull})}|]\right]$$

$$\leq \mathbb{P}\left[|I_{t+1}^{(\text{pull})}| > \left(1 + \frac{27(c+1)\,k\log n}{\mathbb{E}[|I_{t+1}^{(\text{pull})}|]}\right)\mathbb{E}[|I_{t+1}^{(\text{pull})}|]\right] \leq \exp\left(-\frac{\left(\frac{27(c+1)\,k\log n}{\mathbb{E}[|I_{t+1}^{(\text{pull})}|]}\right)^2}{2 + \frac{27(c+1)\,k\log n}{\mathbb{E}[|I_{t+1}^{(\text{pull})}|]}} \cdot \mathbb{E}[|I_{t+1}^{(\text{pull})}|]\right)$$

$$\leq \exp\left(-(27/7)(c+1)\,k\log n\right) \leq n^{-(c+1)}.$$ ◀

C On Small-Set Vertex Expanders with Expansion Larger than 1

As previously discussed, when the parameter $\alpha < 1/2$ we talk about small-set expanders. These graphs are substantially different from the classical notion of expanders, where $\alpha = 1/2$. In fact, if $\alpha = 1/2$ it holds that the vertex-expansion of the graph is bounded, having $\phi \leq 1$. The following lemma formalizes this fact.

▶ **Lemma 18.** *Let $\phi > 0$. There are no (ϕ, α)-expanders for $\alpha > \frac{1}{1+\phi}$.*

Proof. Let S be any set of size $\frac{n}{1+\phi} < |S| \leq \alpha n$. We demand that $\phi(S) \geq \phi$, in other words:

$$|S \cup N(S)| = |S| + |\partial S| \geq |S| + \phi|S| > (1+\phi)\tfrac{n}{1+\phi} = n,$$

which clearly is not possible. ◀

Now the follow-up question is: why do we not simply take $\alpha = \frac{1}{1+\phi}$? The answer is that this is a very restricted graph class as we show next. Recall that the neighborhood of S is defined as $N(S) := \{v \in V : \exists s \in S, \{s,v\} \in E\}$. We now define the *inclusive neighborhood* of a set S as $N[S] := S \cup N(S)$. We define the i-th neighborhood as $N^i[v] := \{u \in V : d(u,v) \leq i\} = N[N^{i-1}[v]]$ where $d(u,v)$ is the shortest-path distance between u and v.

▶ **Lemma 19.** *Let $\phi \geq 1$. If G is a $\left(\phi, \frac{1}{1+\phi}\right)$-expander, then G has diameter at most 2 and minimum degree $\frac{\phi}{1+\phi}n$.*

Proof. Let D denote the diameter of G. Let $u, v \in V$ be a pair of nodes with distance D. Then we have that $u \in N^D[v] \backslash N^{D-1}[v]$. That means that $N^{D-1}[v] \neq V$, so $|N^{D-2}[v]| < \frac{n}{1+\phi}$, and $|N^D[v] \setminus N^{D-1}[v]| < \frac{n}{1+\phi}$. That means that

$$|N^{D-1}[v] \setminus N^{D-2}[v]| = n - |N^{D-2}[v]| - |N^D[v] \setminus N^{D-1}[v]| > n - 2\tfrac{n}{1+\phi} = (\phi - 1)\tfrac{n}{1+\phi}.$$

But that means that the nodes at distance $D-1$, i.e., $N^{D-1}[v] \setminus N^{D-2}[v]$, have edges to the entire graph, so also to v. Hence $N^{D-1}[v] \setminus N^{D-2}[v] = N[v] \setminus v$, and thus $D = 2$.

For the minimum degree we now note that

$$\deg(v) = |N(v)| = n - |N^2[v] \setminus N[v]| - 1 \geq n - \tfrac{n}{1+\phi} = \tfrac{\phi}{1+\phi}n,$$

since $|N^2[v] \setminus N[v]| < \frac{n}{1+\phi}$, so $|N^2[v] \setminus N[v]| + 1 \leq \frac{n}{1+\phi}$. ◀

So to make the definition less restrictive, we allow for α to take more values. However, for small values of α, the expansion property becomes local, which does not help us for rumor spreading.

▶ **Lemma 20.** *Let $\alpha \leq \frac{1}{2+2\phi}$. For any even $n \in \mathbb{N}$, there exists a (ϕ, α)-expander G that is not connected.*

Proof. Let $G = K_{n/2} + K_{n/2}$ be the sum of two disjoint complete graphs, which is clearly not connected. We show that G is a (ϕ, α)-expander. Let $S \subset V$ with $|S| \leq \alpha n$. We have

$$|\partial S| \geq \frac{n}{2} - |S| \geq \frac{n}{2} - \alpha n \geq \frac{n}{2} - \frac{n}{2+2\phi} = \frac{n}{2}\left(1 - \frac{1}{1+\phi}\right) = \frac{\phi n}{2+2\phi} \geq \phi\alpha n \geq \phi|S|. \quad ◀$$

We conclude that for rumor spreading we can focus on the regime $\frac{1}{2+2\phi} < \alpha \leq \frac{1}{1+\phi}$. We show that in this regime a (ϕ, α)-expander has small diameter.

▶ **Lemma 21.** *If G is a (ϕ, α)-expander for $\alpha > \frac{1}{2+2\phi}$, then it has diameter at most $O(\log_\phi n)$.*

Proof. Let $v \in V$ be an arbitrary node, we show that $|N^{\lceil \log_\phi n \rceil}[v]| > n/2$, hence for any pair of nodes $u, v \in V$, we have $N^{\lceil \log_\phi n \rceil}[u] \cap N^{\lceil \log_\phi n \rceil}[v] \neq \emptyset$, so there is a path from u to v of length at most $2\lceil \log_\phi n \rceil$.

By expansion we have that $|N^i[v]| \geq \phi^i$, if $|N^{i-1}[v]| \leq \alpha n$. Now if $|N^{\lceil \log_\phi n \rceil}[v]| \geq \phi^{\log_\phi n} = n$, we are done. So suppose not, i.e., suppose $|N^{\lceil \log_\phi n \rceil - 1}[v]| > \alpha n$. For every $S \subset N^{\lceil \log_\phi n \rceil - 1}[v]$ with $|S| = \alpha n$ it holds that $|S \cup \partial S| \geq (1+\phi) \cdot \alpha n$ since G is a (ϕ, α)-expander. Hence $|N^{\lceil \log_\phi n \rceil}[v]| \geq |S \cup \partial S| \geq (1+\phi) \cdot \alpha n$. Since we assume $\alpha > \frac{1}{2+2\phi}$ we have that $(1+\phi)\alpha n > \frac{1+\phi}{2+2\phi}n = \frac{n}{2}$, which completes the proof. ◀

D Examples of Small-Set Vertex Expanders

A trivial example of (ϕ, α)-expanders are complete graphs.

▶ **Lemma 22.** *The complete graph on n nodes is a (ϕ, α)-vertex expander for every $0 < \phi \leq n - 1$ and $\alpha \leq \frac{1}{1+\phi}$.*

However, we can show also the existence of significantly sparser graphs that are (ϕ, α)-expanders. We consider Erdős-Rényi random graphs $G(n, p)$ on n nodes, where there is an edge between any pair of nodes with probability p. We also know that w.h.p. the graphs $G(n, \phi/n)$ have diameter $\Theta(\log_\phi n)$ for $\phi = \omega(1)$ [15].

▶ **Lemma 23.** *Let $\alpha < \frac{1}{1+\phi/(1-e^{-1})} \approx \frac{1}{1+1.58\phi}$. Let a be such that $\alpha = \frac{1}{1+a\phi}$ and $\phi \geq \Theta\left(\left(\frac{1}{a(1-e^{-1})-1}\right)^2 \log n\right)$. Erdős-Rényi random graphs $G(n, p)$ with $p \geq \frac{3\phi}{n}$ are (ϕ, α)-vertex expanders with high probability.*

On the h-Majority Dynamics with Many Opinions

Francesco d'Amore
Gran Sasso Science Institute, L'Aquila, Italy

Niccolò D'Archivio
INRIA, COATI, Université Côte d'Azur, Nice, France

George Giakkoupis
INRIA Rennes, France

Emanuele Natale
CNRS, I3S, COATI, Université Côte d'Azur, Nice, France

Abstract

We present the first upper bound on the convergence time to consensus of the well-known h-majority dynamics with k opinions, in the synchronous setting, for h and k that are both non-constant values. We suppose that, at the beginning of the process, there is some initial *additive bias* towards some plurality opinion, that is, there is an opinion that is supported by x nodes while any other opinion is supported by strictly fewer nodes. We prove that, with high probability, if the bias is $\omega(\sqrt{x})$ and the initial plurality opinion is supported by at least $x = \omega(\log n)$ nodes, then the process converges to plurality consensus in $O(\log n)$ rounds whenever $h = \omega(n \log n/x)$. A main corollary is the following: if $k = o(n/\log n)$ and the process starts from an almost-balanced configuration with an initial bias of magnitude $\omega(\sqrt{n/k})$ towards the initial plurality opinion, then any function $h = \omega(k \log n)$ suffices to guarantee convergence to consensus in $O(\log n)$ rounds, with high probability. Our upper bound shows that the lower bound of $\Omega(k/h^2)$ rounds to reach consensus given by Becchetti et al. (2017) cannot be pushed further than $\widetilde{\Omega}(k/h)$. Moreover, the bias we require is asymptotically smaller than the $\Omega(\sqrt{n \log n})$ bias that guarantees plurality consensus in the 3-majority dynamics: in our case, the required bias is at most any (arbitrarily small) function in $\omega(\sqrt{x})$ for any value of $k \geq 2$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Probabilistic algorithms; Mathematics of computing → Markov processes; Applied computing → Systems biology; Networks → Network protocols

Keywords and phrases Distributed Algorithms, Randomized Algorithms, Markov Chains, Consensus Problem, Opinion dynamics, Plurality Consensus

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.27

Related Version *Full Version*: `https://arxiv.org/abs/2506.20218` [17]

Funding This work was partially supported by the MUR (Italy) Department of Excellence 2023 - 2027 for GSSI, the AID INRIA-DGA project n°2023000872 "BioSwarm", and the 3IA Côte d'Azur Investments in the Future project ANR-19P3IA-0002.

Acknowledgements The authors are grateful to Frédéric Giroire for helpful discussions regarding this project.

1 Introduction

In this paper, we study the convergence time of the well-known h-majority dynamics. In brief, the setting is the following: We are given a network of n agents that interact with each other in synchronous rounds. At round 0, each agent supports one out of k possible opinions in the set $[k]$: At round $i > 0$, each agent v samples uniformly at random h neighbors $u_1, \ldots, u_h$ (with repetition) which, in turn, send their opinions $x_1, \ldots, x_h$ to v. Then, v adopts as its own opinion the mode of $\{x_1, \ldots, x_h\}$, breaking ties uniformly at random. Let


© Francesco d'Amore, Niccolò D'Archivio, George Giakkoupis, and Emanuele Natale;
licensed under Creative Commons License CC-BY 4.0



$\mathbf{c}_t(i)$ count the number of agents supporting opinion i after round number t. The h-majority dynamics gives rise to a random process on the system graph, which we call the h-majority process. The question is to determine the number of communication rounds that the process needs to reach *consensus*, i.e., a configuration where all agents agree on some opinion $i \in [k]$. Furthermore, we are interested in *plurality consensus*: Suppose that at the beginning of the process there is a *plurality opinion*, that is, an opinion $i^\star$ such that $\mathbf{c}_0(i^\star) > \mathbf{c}_0(j)$ for all $j \neq i^\star$. Plurality consensus is consensus on the opinion $i^\star$, and one wishes to quantify the size of the initial bias that guarantees to obtain plurality consensus with high probability.[1]

1.1 Our contribution

We give the first non-trivial upper bound to the consensus time of the h-majority process with k opinions for non-constant h and k. Let us denote the difference $\max_i \min_{j \neq i}\{\mathbf{c}_t(i) - \mathbf{c}_t(j)\}$ by B_t and refer to it as the *bias* of the configuration $\mathbf{c}_t$ at time t. In a nutshell, our main result is the following.

▶ **Theorem 1.** *Consider the h-majority dynamics with k opinions in the complete graph of n nodes with self-loops* [2]*. Let $\lambda_1, \lambda_2, \lambda_3$ be large enough positive constants. Assume that opinion 1 is the initial plurality opinion with $\mathrm{B}_0 \geq \lambda_1 \sqrt{\mathbf{c}_0(1)}$ and $\mathbf{c}_0(1) \geq \lambda_2 \log n$. If $h \geq \lambda_3 n \log n / \mathbf{c}_0(1)$, then, with high probability, the h-majority process reaches plurality consensus within $O(\log n)$ rounds.*

Let us discuss Theorem 1 a bit more in details. First, notice that it always holds $\mathbf{c}_0(1) \geq n/k$ since $k \geq n/\mathbf{c}_0(1)$. In the following, we say that a configuration is almost-balanced if there exists a subset $I \subseteq [k]$, with $|I| = O(1)$, such that $\mathbf{c}_0(i) = \Theta(\mathbf{c}_0(1))$ for all $i \in [k] \setminus I$, assuming that opinion 1 is the plurality opinion. Notice that, in this case, it must hold that $\mathbf{c}_0(1) = \Theta(n/k)$ and $\mathbf{c}_0(i) = \Theta(n/k)$ for all $i \in [k] \setminus I$.

Case 1: $k = O(n/\log n)$.

Here we have two sub-cases depending on how we fix $\mathbf{c}_0(1)$. In the first sub-case, we fix $\mathbf{c}_0(1) = \Theta(n/k)$ such that $\mathbf{c}_0(1) \geq \lambda_2 \log n$ (which is always possible). Here, we must select some large enough $h = \Theta(n \log n / c_0(1)) = \Theta(k \log n)$ in order to apply Theorem 1, with the required bias that is minimum, that is, $\mathrm{B}_0 = \Theta(\sqrt{n/k})$: observe that the configuration is necessarily almost-balanced. If, instead, $\mathbf{c}_0(1) = \omega(n/k)$, it is guaranteed that $\mathbf{c}_0(1) = \omega(\log n)$. In this case, the configuration *cannot* be almost-balanced because the required bias is $\mathrm{B}_0 = \Theta(\sqrt{\mathbf{c}_0(1)}) = \omega(\sqrt{n/k})$. Here, we can pick $h = \Theta(n \log n / \mathbf{c}_0(1)) = o(k \log n)$.

Case 2: $k = \omega(n/\log n)$.

Since $k = \omega(n/\log n)$ and $\mathbf{c}_0(1) = \Omega(\log n)$ by hypothesis, we *cannot* have almost balanced configurations: we can always choose some $h = o(k \log n)$ and we will have that the minimum bias is $\mathrm{B}_0 = \omega(\sqrt{n/k})$. Note that in this setting, h may be larger than n, which we do not exclude as the samples can contain repetitions.

---

[1] The expression *with high probability* (in short, w.h.p.) refers to an event that holds with probability at least $1 - n^{-c}$, where $c > 0$ is any constant that is independent of the size n of the distributed system.

[2] Assuming self-loops is standard in the literature, as it simplifies the analysis without affecting the possible outcomes.

Observations

Despite the assumption $h \geq \lambda_3 n \log n / \mathbf{c}_0(1)$ is undoubtedly a strong one (that is, h needs to be at least roughly $k \log n$ for almost-balanced configurations), in Section 3 we show how the analysis for such regime of h is already quite involved and we discuss what the main difficulties to reduce such assumption are. Also, in Section 3 we will highlight why we believe that $h \sim n/\mathbf{c}_0(1)$ is a "threshold" value, since the two settings $h \ll n/\mathbf{c}_0(1)$ and $h \gg n/\mathbf{c}_0(1)$ seem to need complementary approaches, which is also why our analysis cannot be adapted to the case $h \ll n/\mathbf{c}_0(1)$. However, since h is a "static" parameter of the h-majority dynamics, in order to approach the case $h < \lambda_3 n \log n/\mathbf{c}_0(1)$, one has to show only that there is a time $t > 0$ such that $\mathbf{c}_t(1) \geq \lambda_3 n \log n/h$ and that opinion 1 is still the plurality opinion with B_t that meets the hypothesis of Theorem 1, which implies that our analysis applies.

Prior to this work, the only result for the h-majority dynamics with non-constant h and k was a lower bound of $\Omega(k/h^2)$ rounds to reach consensus given by [5] that holds with high probability whenever $\max_{i\in[k]}\{\mathbf{c}_0(i)\} \leq 3n/(2k)$. Hence, Theorem 1 shows that the lower bound cannot be pushed to $\widetilde{\Omega}(k/h)$ (where the notation $\widetilde{\Omega}$ hides polylog(n) factors), at least as long as $\max_{i\in[k]}\{\mathbf{c}_0(i)\} \leq 3n/(2k)$ is satisfied, e.g., when $(n - \mathrm{B}_0)/(\mathbf{c}_0(1) - \mathrm{B}_0) \leq k \leq 3n/(2\mathbf{c}_0(1))$, which is always realized when $\max\{(n+(k-1)\mathrm{B}_0)/k, \lambda_2 \log n\} \leq \mathbf{c}_0(1) \leq 3n/(2k)$ and $k \leq 3n/(2\lambda_2 \log n)$ for any value of B_0 meeting the hypothesis of Theorem 1.

Furthermore, as we will discuss in the related works, in the context of opinion dynamics, the minimum required bias in any other opinion dynamics is always asymptotically larger than our minimum required bias. Interestingly, the h-majority dynamics can amplify a very small bias even when $h = O(k \log n)$ (this is easy to prove for very large h as we argue in Section 3, but it is not at all trivial for our parameters).

A final remark that we stress is about the message complexity. According to Theorem 1, the message complexity required to reach consensus is $O(hn \log n)$, which can be as high as $O(kn \log^2 n)$ in almost-balanced configurations. By recent work on the 3-majority dynamics (see Section 1.2), the message complexity of the 3-majority dynamics is $O(kn \log n)$. We believe that the gap between these two quantities is due to the fact that the analysis of the h-majority dynamics is not yet tight.

1.2 Previous results

There is a vast body of research that investigated the h-majority dynamics when either h or k are constant values, both in the synchronous and in the asynchronous settings [5,9,10,13,27,34]. Before exploring the results in the two settings, let us discuss the work [10]. The authors compared the power of the h-majority dynamics and that of the $(h+1)$-majority dynamics with $k = 2$ opinions. They proved that, starting from the same configuration, the consensus time T_{h+1} of the $(h+1)$-majority dynamics is stochastically dominated by the consensus time T_h of the h-majority dynamics in the following sense: for each $t > 0$, $\Pr(T_{h+1} \leq t) \geq \Pr(T_h \leq t)$ in both the synchronous and the asynchronous settings. However, to date no such extension to the case of $k > 2$ opinions is known.

Synchronous setting

Let us specify that all the results we mention in this subsection and the next two ones hold in the complete graph of n nodes with self-loops. The work [6] proved an upper bound of $O\left((k^2\sqrt{\log n} + k \log n)(k + \log n)\right)$ rounds to reach consensus that holds w.h.p., provided that $k \leq n^\alpha$ for a suitable positive constant $\alpha < 1$.

In [5], the authors showed that the synchronous 3-majority dynamics with k opinions converges in $O(\min\{k, (n/\log n)^{\frac{1}{3}}\}\log n)$ with high probability, provided that the bias of the initial configuration is at least $c\sqrt{\min\{2k, (n/\log n)^{\frac{1}{3}}\}n\log n}$ for some constant $c > 0$. Moreover, the authors provided a lower bound of $\Omega(k\log n)$ on the convergence time to consensus, w.h.p., whenever the initial configuration is sufficiently balanced, that is, $\max_{i\in[k]}\{\mathbf{c}_0(i)\} \leq n/k + (n/k)^{1-\varepsilon}$ for some $\varepsilon > 0$ and $k \leq (n/\log n)^{1/4}$.

The work [9] compared the synchronous 3-majority dynamics with the synchronous 2-choices dynamics. The 2-choices dynamics works as follows: At each round, each agent picks two neighbors (with repetition) u.a.r. If the two sampled neighbors support the same opinion, the agent adopts that opinion. Otherwise, the agent keeps its own opinion. The authors first proved a generic lower bound of $\Omega(\min\{k, n/\log n\})$ rounds to reach consensus starting from the initial perfectly balanced configuration, w.h.p. Furthermore, they proved that the 3-majority dynamics works better in symmetric configurations (i.e., with no initial bias) when, e.g., $\max_{i\in[k]}\{\mathbf{c}_0(i)\} = O(\log n)$. In particular, w.h.p., the 3-majority takes time at most $O(n^{3/4}\log^{7/8} n)$ to reach consensus w.h.p., regardless of any other hypothesis on the initial configuration, while the 2-choices needs time $\Omega(n/\log n)$ whenever $\max_{i\in[k]}\{\mathbf{c}_0(i)\} = O(\log n)$. This was the first work to notice that, for a large number of opinions, the 3-majority dynamics is polynomially (in k) faster than the 2-choices dynamics.

The work [27] improved upon [5] and showed that, for the 2-choices with $k = O(\sqrt{n/\log n})$ and for the 3-majority with $k = O(n^{1/3}/\sqrt{\log n})$, the convergence time to consensus is $O(k\log n)$, with high probability. Notice that this upper bound is tight according to the lower bound by [5], at least as long as $k \leq (n/\log n)^{1/4}$. Furthermore, the authors showed that the convergence time on the 3-majority dynamics is $O(n^{2/3}\log^{3/2} n)$ with high probability, regardless of the number of opinions.

Very recently, [34] settled almost tightly the complexity of both the 3-majority and the 2-choices dynamics. The authors proved that, w.h.p., the 3-majority dynamics reaches consensus in $O(k\log n)$ rounds whenever $k = o(\sqrt{n}/\log n)$, while it takes time $O(\sqrt{n}\log^2 n)$ for other values of k. Furthermore, [34] proved that plurality consensus is ensured w.h.p. as long as the initial bias is $\mathrm{B}_0 = \omega(\sqrt{n\log n})$. As for the 2-choices, they showed that, w.h.p., the dynamics reaches consensus in $O(k\log n)$ rounds whenever $k = o(n/\log^2 n)$, while it takes time $O(n\log^3 n)$ otherwise. In this case, plurality consensus is ensured w.h.p. as long as the initial bias is $\mathrm{B}_0 = \omega(\sqrt{\mathbf{c}_0(1)\log n})$.[3] These results almost match the generic lower bound given by [9], up to logarithmic factors.

Asynchronous setting

The only works that analyzed the 3-majority dynamics in the asynchronous setting are [10,13]. In [10], the authors consider the binary opinion case and show that the convergence time of the asynchronous 3-majority dynamics is $O(n\log n)$ rounds, w.h.p., and that a bias of $\Theta(\sqrt{n\log n})$ is sufficient to ensure plurality consensus, w.h.p. The authors of [13] showed that the convergence time is $O(\min\{kn\log^2 n, n^{3/2}\log^{3/2} n\})$, w.h.p., no matter the number of initial opinions. They also provided a generic lower bound of $\Omega(\min\{kn, n^{3/2}/\sqrt{\log n}\})$ rounds to reach consensus (starting from balanced configurations), w.h.p. The work [13] (which came before [34]) was the first to establish exactly how the linear-in-k dependence in

[3] This is the only example of initial bias that gets "close enough" to what we require in Theorem 1, but with an exceeding $\sqrt{\log n}$ factor.

the consensus time of the 3-majority dynamics breaks when the number of opinions grows over $\sqrt{n}$, in which case the consensus time is sublinear in the number of opinions. The reader may observe that the asynchronous setting has exactly the same convergence time as the synchronous model, but for a multiplicative factor n. This is to be expected: In the asynchronous setting, in a round only one agent (sampled u.a.r.) updates its state, and hence we need roughly n rounds to activate all agents at least once (up to polylogarithmic factors). For processes with small variance (smaller w.r.t. the voter model), convergence times from the synchronous to the asynchronous setting usually scale with such a factor. Note that this is not true for processes with large variance [7].

On the h-majority dynamics

As previously discussed, [10] established the "hierarchy" of the h-majority dynamics when the number of opinions is $k = 2$, by showing that the $(h + 1)$-majority is "faster" than the h-majority (both in the synchronous and asynchronou settings), and [5] exhibited the lower bound of $\Omega(k/h^2)$ rounds to reach consensus (that holds w.h.p.). Apart for the two aforementioned works, there is no theoretical result on the h-majority dynamics for $h \gg 1$, which makes it one of the open problems in consensus dynamics. [34] put the h-majority dynamics in the "Open Question" section of their recent work, and wondered whether the techniques developed in [34] applied to the h-majority dynamics as well. We do not know the answer to this question: As we will discuss in Section 3, in the h-majority the main difficulty is to compute the expectation of $\mathbf{c}_{t+1}(i)$ conditional on the whole configuration $\mathbf{c}_t$ at the previous rounds, while this is straightforward when $h = 3$ and is at the core of every paper analyzing the 3-majority dynamics [5, 6, 13, 27, 34]. More in general, there is no non-trivial upper bound to the consensus time of the h-majority dynamics with $k \gg 1$ opinions. In this paper, we take a first step into the analysis of the h-majority dynamics. As we will show, our main effort basically is providing a lower bound to $\mathbb{E}[\mathbf{c}_{t+1}(1) - \mathbf{c}_{t+1}(2) \mid \mathbf{c}_t]$, supposing that opinion 1 is the plurality opinion at round t, and opinion 2 is the opinion supported by the second-largest community.

Other related works

The 3-majority, the h-majority, and other majority-based dynamics have been investigated also in other settings. For example, in presence of communication noise (which corrupts the exchanged messages) or in presence of stubborn agents [18, 19], or when some opinion is preferred among the others, that is, there is some probability that an agent, instead of running the protocol, spontaneously switches to the preferred opinion with some fixed probability [14, 32].

Other opinion dynamics for the (plurality) consensus problem have been investigated both in the synchronous and asynchronous settings. One of the most prominent is the undecided-state dynamics. In the undecided-state dynamics there is an extra opinion, the *undecided* opinion. The update rule works as follows: A node samples one neighbor u.a.r. and pulls its opinion. If the received opinion is different from the supported one, the node becomes undecided. If the node is undecided, it just copies whatever opinion receives. The undecided-state dynamics has been studied both in the synchronous setting and in the population protocol model (asynchronous setting) [1–4, 8, 12, 15, 16] and performs roughly the same as the 3-majority dynamics for a small number of opinions (while its analysis for the unconstrained general case is still open). More in general, work on opinion dynamics fits into the recent trend in the distributed computing community of drawing inspiration

from social and biological systems. Other then the consensus problem, researchers have analyzed distributed search problems but also more generic problems (MIS, leader election, etc.) in extremely weak computational models (such as the beeping model or the stone-age model) [11, 20, 22–24, 26, 28, 29, 31, 35].

A special mention goes to [25]. Here, the authors considered noisy distributed systems, i.e., systems in which exchanged messages can be corrupted and changed with some positive probability, a setting that takes inspiration by biological societies, where environmental factors can disturb communication. They characterized the type of noise for which the tasks of *information spreading* and plurality consensus can be solved, when the number of opinions is constant.[4] The protocol designed by [25] to solve plurality consensus relies on some majority-based rule, and proves a technical result that is at the heart of our analysis. We will discuss this in more details in Section 3. We nevertheless stress the fact that [25] only considers the case where the number of opinions k is a constant.

2 Preliminaries

We work on a complete graph of n nodes with self-loops. Initially, each node supports one out of k opinions in the set $[k]$. Time is synchronous and controlled by some global clock. At each round, in parallel, every node v samples h neighbors u.a.r. with repetition and pulls their opinions. Then, v adopts the unique majority opinion, if any, breaking ties u.a.r. among the opinions that are in the majority. For each opinion $i \in [k]$, we denote by $\mathbf{c}_t(i)$ the number of nodes supporting opinion i at time after performing the update-rule at time t. Let $\mathbf{c}_t = (\mathbf{c}_t(1), \ldots, \mathbf{c}_t(k))$ denote the *configuration* of the system at time t. Notice that the random variable $\mathbf{c}_t$ defines a Markov chain: for any sequence of configurations $y_1, \ldots, y_t \in [n]^k$, it holds that $\Pr\big(\mathbf{c}_t = y_t \mid \cap_{i=1}^{t-1}\{\mathbf{c}_i = y_i\}\big) = \Pr(\mathbf{c}_t = y_t \mid \mathbf{c}_{t-1} = y_{t-1})$.

Without loss of generality, we assume that at the beginning of the process we have $\mathbf{c}_0(1) \geq \mathbf{c}_0(2) \geq \ldots \mathbf{c}_0(k)$. At any time t, we call *plurality opinion* at time t the opinion i such that $\mathbf{c}_t(i) \geq \mathbf{c}_t(j)$ for all $j \in [k]$. At any time t, the *bias* of a configuration $\mathbf{c}_t$ is the quantity $\mathrm{B}_t = \max_{i\in[k]} \min_{j\neq i}\{\mathbf{c}_t(i) - \mathbf{c}_t(j)\}$, that is, of how many nodes the community supporting the plurality opinion exceeds the community supporting the second largest one. Note that $\mathrm{B}_t = 0$ if there are multiple plurality opinions, while $\mathrm{B}_t > 0$ the plurality opinion is unique.

We work from the perspective of a single node. When a node v samples h neighbors u.a.r., we are drawing from a multinomial distribution. In particular, given a configuration $\mathbf{c}_t$ at time t and a node v, let $\mathbf{X}^{(t)}(v) = (X_1^{(t)}(v), \ldots, X_k^{(t)}(v))$ be a multinomial random variable where $X_i^{(t)}(v) \sim \mathrm{Bin}(n, p_i^{(t)})$, with $p_i^{(t)} = \mathbf{c}_t(i)/n$, counts the number of neighbors supporting opinion i that v samples at time t: it must hold that $\sum_{i=1}^{k} X_i^{(t)}(v) = n$, so the variables are negatively correlated. Let us denote the event that v supports opinion i at the end of round t by $\mathcal{W}_i^{(t)}(v)$.

For the rest of the paper, we will use only $p_1^{(t)}, \ldots, p_k^{(t)}$, since these are the probabilities defining the multinomial distribution of interest. We define the *normalized bias* to be $\delta_t = \max_{i\in[k]} \min_{j\neq i}\{p_i^{(t)} - p_j^{(t)}\}$, which is equal to B_t/n. When the round we are referring to is clear from the context, we only write $p_1, \ldots, p_k$, δ, $\mathbf{X}(v) = (X_1(v), \ldots, X_k(v))$, and $\mathcal{W}_i(v)$, omitting the dependency on t, to refer to $p_1^{(t)}, \ldots, p_k^{(t)}$, δ_t, $\mathbf{X}^{(t)}(v) = (X_1^{(t)}(v), \ldots, X_k^{(t)}(v))$,

[4] Information spreading is defined as follows: In the distributed system there is only one node that is informed about one out k opinions, while all other nodes are not informed. The task is to design a protocol that informs all nodes as fast as possible.

and $\mathcal{W}_i^{(t)}(v)$. Also, since in any given round t the random variables $\{\mathbf{X}(v)\}_{v\in V}$ are i.i.d., and the events $\{\mathcal{W}_i(v)\}_{v\in V}$ are mutually independent, when analyzing the distribution of some $\mathbf{X}(v)$ or the probabilities of some $\mathcal{W}_i(v)$, we omit the dependency on v and only write $\mathbf{X} = (X_1, \ldots, X_k)$ and $\mathcal{W}_i$.

From now on, we will refer to the normalized bias δ_t simply as the *bias* of the configuration, as we will not make use of B_t anymore. With this notation, our main theorem can be rewritten as follows:

▶ **Theorem 2.** *Let $\lambda_1, \lambda_2, \lambda_3$ be large enough positive constants. Assume that opinion 1 is the initial plurality opinion with $\delta_0 \geq \lambda_1\sqrt{p_1/n}$ and $p_1 \geq \lambda_2 \log n/n$. If $hp_1 \geq \lambda_3 \log n$, then, with high probability, the h-majority process reaches plurality consensus within $O(\log n)$ rounds.*

3 Overview of our analysis and technical challenges

The whole analysis is an effort to estimate the quantity $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$ from below (assuming that $p_1 \geq p_2 \geq \ldots \geq p_k$), and use it to get a lower bound on the expect round-by-round growth of the bias δ. Notice that, for any given configuration $\mathbf{c}_t$ at time t, it holds that

$$\begin{aligned}\mathbb{E}[\delta_{t+1} \mid \mathbf{c}_t] &\geq \mathbb{E}\left[\max_{i\in[k]}\min_{j\neq i}\left\{\Pr\left(\mathcal{W}_i^{(t+1)}\right) - \Pr\left(\mathcal{W}_j^{(t+1)}\right)\right\} \;\middle|\; \mathbf{c}_t\right] \\ &\geq \mathbb{E}\left[\min_{j\neq 1}\left\{\Pr\left(\mathcal{W}_1^{(t+1)}\right) - \Pr\left(\mathcal{W}_j^{(t+1)}\right)\right\} \;\middle|\; \mathbf{c}_t\right] \\ &\geq \mathbb{E}\left[\Pr\left(\mathcal{W}_1^{(t+1)}\right) - \Pr\left(\mathcal{W}_2^{(t+1)}\right) \;\middle|\; \mathbf{c}_t\right]. \qquad (1)\end{aligned}$$

Suppose that $p_1^{(t)} = p_2^{(t)} + \delta_t > p_2^{(t)} \geq \ldots \geq p_k^{(t)}$. This implies that $\Pr(\mathcal{W}_1^{(t+1)} \mid \mathbf{c}_t) - \Pr(\mathcal{W}_i^{(t+1)} \mid \mathbf{c}_t) \geq \Pr(\mathcal{W}_1^{(t+1)} \mid \mathbf{c}_t) - \Pr(\mathcal{W}_2^{(t+1)} \mid \mathbf{c}_t)$ for all $i \geq 3$ since $p_2^{(t)} \geq p_i^{(t)}$ for $i \geq 3$. If we proved that $\Pr(\mathcal{W}_1^{(t+1)} \mid \mathbf{c}_t) - \Pr(\mathcal{W}_2^{(t+1)} \mid \mathbf{c}_t) \geq \delta_t(1+x)$ for some constant $x > 0$, then we would get $\mathbb{E}[\delta_{t+1} \mid \mathbf{c}_t] \geq \delta_t(1+x)$, which allows us to concentrate and establish that $\delta_{t+1} \geq \delta_t(1 + x/2)$ w.h.p. (conditional on $\mathbf{c}_t$), for a suitable choice of $x > 0$ and for a large enough δ_t, thanks to the Hoeffding bound (Lemma 23).

Notice that, in general, the probability of $\mathcal{W}_i$ is the probability that X_i is either the unique maximum or, if it is a maximum and is not unique, it must win the tie broken u.a.r. In formulas, the probability is

$$\Pr(\mathcal{W}_i) = \sum_{m=1}^{k} \frac{1}{m} \sum_{\substack{1\leq i_1<\ldots<i_m\leq k:\\ i_j\neq i\,\forall j\in[m]}} \Pr\left(\bigcap_{j\in[m]}\{X_i = X_{i_j}\} \cap \Big(\bigcap_{j\notin\{i,i_1,\ldots,i_m\}}\{X_i > X_j\}\Big)\right). \qquad (2)$$

Using directly Equation (2) to bound $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$ is difficult, and we did not find general estimations in the literature.

Trivial case: very large h

If, e.g., h is very large compared to δ, in one round all nodes would adopt the plurality opinion with high probability. The reason follows.

Suppose $p_1 = p_2 + \delta > p_2 \geq \ldots \geq p_k$. Then, by the multiplicative Chernoff bound (Lemma 22 in Section B) we get that

$$\Pr(X_1 \geq hp_1(1-x)) \geq 1 - \exp\big(-\Theta(x^2 hp_1)\big).$$

Similarly,

$$\Pr(X_i \leq hp_2(1+x)) \geq 1 - \exp\big(-\Theta(x^2 hp_2)\big).$$

Suppose p_1 and p_2 are comparable, that is, $p_1 = \Theta(p_2)$ (anyway the bias we require is always an $o(p_1)$). Then, $X_1 \geq hp_1(1-x)$ and $X_i \leq hp_2(1+x)$ for all $i \geq 2$ hold w.h.p. as long as $h \geq C \log n/(p_1 x^2)$ for a large enough constant $C > 0$. If $hp_1(1-x) > hp_2(1+x)$, then $\mathcal{W}_1$ holds w.h.p., that is, a node would adopt opinion 1 in just 1 round w.h.p. Playing with constants, by the union bound, one would obtain plurality consensus w.h.p. in just 1 round. The condition is satisfied whenever $\delta > x(p_1 + p_2)$, that is, for $x < \delta/(p_1 + p_2)$. Since $h \geq C \log n/(p_1 x^2)$ and $x < \delta/(p_1 + p_2)$ must hold at the same time, it is sufficient to choose $h \geq C' p_1 \log n/\delta^2$ for a large enough constant $C' > C$. Notice that, since our bias can be such that $\delta \sim \sqrt{p_1/n}$, we must have that $hp_1 > (C'/\lambda_1^2)p_1 \cdot n \log n \gg \log n$ for any choice of $p_1 = \Omega(\log n/n)$. The minimum that h can be with this approach is $\Theta(\log^2 n/p_1)$ and can be as large as $h = \Omega(n^{2/3}/p_1)$ if $p_1 \geq 1/n^{1/3}$. Our Theorem 2 guarantees us that we can always set $h = \Theta(\log n/p_1)$ as long as $p_1 \geq \lambda_2 \log n/n$ for a large enough constant $\lambda_2 > 0$. For the minimum bias $\delta = \lambda_1\sqrt{p_1/n}$, we have that $p_2 = p_1 - o(p_1)$. Hence, $\mathbb{E}[X_2] = hp_2 = hp_1 - o(hp_1) = \mathbb{E}[X_1] - o(\log n)$. Unfortunately, we cannot capture such a deviation from the average of X_1 using concentration bounds, so we need to adopt a different approach.

3.1 Our approach

At the heart of our analysis, there is a technical lemma that was proved in [25, Lemma 9].

The lemma can be reformulated as follows. Suppose we are performing the h-majority when in the system there are only $k = 2$ opinions. Then, [25, Lemma 9] provides a tight lower bound on the expected bias after one round, i.e., on $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$. Notice that, when $k = 2$, $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2) = \Pr(X_1 > X_2) + \Pr(X_1 = X_2)/2 - \Pr(X_2 > X_1) - \Pr(X_1 = X_2)/2$, which is equal to $\Pr(X_1 > X_2) - \Pr(X_2 > X_1)$.

▶ **Lemma 3** (Lemma 9 in [25]). *For any integer $h > 0$, let $X_1 \sim \text{Binomial}(h, p)$ and $X_2 = h - X_1 \sim \text{Binomial}(h, 1-p)$, with $p > 1/2$. Let $\delta = 2p - 1$ be the bias. Then, we have that*

$$\Pr(X_1 > X_2) - \Pr(X_2 > X_1) \geq \sqrt{\frac{2h}{\pi}} \cdot g(\delta, h),$$

where

$$g(\delta, h) = \begin{cases} \delta\left(1-\delta^2\right)^{\frac{h-1}{2}} & \text{if } \delta < \frac{1}{\sqrt{h}}, \\ \frac{1}{\sqrt{h}}\left(1-\frac{1}{h}\right)^{\frac{h-1}{2}} & \text{if } \delta \geq \frac{1}{\sqrt{h}}. \end{cases}$$

We show how to adapt the proof of [25, Lemma 9] to get Lemma 3 in the full version of the paper [17]. Basically, Lemma 3 makes explicit the minimum sample size required by the expectation of the bias in the next round to increase by a constant multiplicative factor relative to the current bias, to which we refer here as δ. More specifically, it is sufficient that the number of samples h satisfies $\sqrt{\frac{2h}{\pi}}\left(1-\delta^2\right)^{\frac{h-1}{2}} \geq \sqrt{\frac{2h}{\pi e}} \geq c$ for some positive constant $c > 0$ as long as $\delta < \frac{1}{\sqrt{h}}$. If, instead, $\delta \geq \frac{1}{\sqrt{h}}$, the lemma states that the new expected value of the bias is more than a constant, which allows us to show that the bias increases just by standard concentration arguments around the averages of X_1 and X_2 (through Berry-Esseen's inequality or Chernoff bounds depending on how large δ is w.r.t. h).

We would like to exploit something that is similar, in spirit, to Lemma 3 when $k \gg 1$, in order to get that the bias increases each round. However, $X_1 \sim \text{Binomial}(h, p_1)$ and $X_2 \sim \text{Binomial}(h, p_2)$ but $p_1 + p_2 < 1$ and $X_2 \neq h - X_1$. In order to overcome this issue, let us define few events.

▶ **Definition 4** (Winning events). *Let $i \in [k]$. We define $\mathcal{W}_{i,\text{ties}}$ to be the event in which opinion "i" is the most sampled opinion, including possible ties. Formally,*

$$\mathcal{W}_{i,\text{ties}} = \cap_{j \in [k] \setminus \{i\}} \{X_i \geq X_j\}. \tag{3}$$

Similarly, we define $\mathcal{W}_{i,\text{strict}}$ to be the event in which opinion "i" is the most sampled opinion, without ties. Formally,

$$\mathcal{W}_{i,\text{strict}} = \cap_{j \in [k] \setminus \{i\}} \{X_i > X_j\}. \tag{4}$$

Finally, for $i \neq j \in [k]$, we define $\mathcal{W}_{i,j} = \mathcal{W}_i \cup \mathcal{W}_j$, $\mathcal{W}_{i,j,\text{strict}} = \mathcal{W}_{i,\text{strict}} \cup \mathcal{W}_{j,strict}$, and $\mathcal{W}_{i,j,\text{ties}} = \mathcal{W}_{i,\text{ties}} \cup \mathcal{W}_{j,ties}$.

Clearly, $\mathcal{W}_{i,\text{strict}} \subseteq \mathcal{W}_i \subseteq \mathcal{W}_{i,\text{ties}}$ and $\mathcal{W}_{i,j,\text{strict}} \subseteq \mathcal{W}_{i,j} \subseteq \mathcal{W}_{i,j,\text{ties}}$. The idea is to perform a conditional analysis: if we condition on $\mathcal{W}_{1,2}$, then we already know that either X_1 or X_2 win at the next round. However, we do not quite get that the conditional random variable $(X_1 \mid \mathcal{W}_{1,2})$ is a binomial one. At the same time, it is not true that $(X_2 \mid \mathcal{W}_{1,2})$ counts the number of failures of $(X_1 \mid \mathcal{W}_{1,2})$. The whole Section 4.1 presents the analytical effort to demonstrate that, in practice, we can get some result that is similar to Lemma 3. The section is quite technical and involved, with ad-hoc results that are needed for this adaptation. We stress that we actually use, as conditional event, the event $\mathcal{W}_{1,2,\text{strict}}$. The reason being that (as we show in Lemma 7)

$$\begin{aligned} &\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}}) \\ =\ &\Pr(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}}), \end{aligned} \tag{5}$$

while the equality is not true if instead of $\mathcal{W}_{1,2,\text{strict}}$ we use $\mathcal{W}_{1,2}$, because we need to take ties into account. Observe that Equation (5) gives a formula that resembles the one that is estimated in Lemma 3.

In Section 4.2, we estimate the probability of $\mathcal{W}_{1,2,\text{strict}}$ with respect to $\mathcal{W}_{1,2}$ and $\mathcal{W}_{1,2,\text{ties}}$. In order to do that, we classify the k opinions in two classes: the *strong* and the *rare* opinions. The strong ones are opinions whose probabilities are comparable to p_1, say, $p_i > p_1/2$. The rare ones are all the others. Through concentration arguments, we can show that rare opinions disappear in one round w.h.p., and it is easy to see, by symmetry arguments, that $\mathcal{W}_1 = \Omega(p_1)$. The main result that we obtain in Section 4.2 is that, basically, under the hypothesis that $hp_1 \geq C \log n$ for some large enough constant $C > 0$, $p_1/8 \leq \Pr(\mathcal{W}_{1,\text{ties}})/8 \leq \Pr(\mathcal{W}_{1,\text{strict}}) \leq \Pr(\mathcal{W}_1) \leq \Pr(\mathcal{W}_{1,\text{ties}})$, that is, *ties do not matter that much.* Notice that the latter fact is, again, trivial if $hp_1 = \Omega(\max\{p_1^2 n, \log^2 n\} \gg O(\log n)$, while we require a smaller (and, possibly, *much* smaller) h. Also, observe that $\Pr(\mathcal{W}_{1,\text{strict}}) \geq p_1/8$ implies that $\Pr(\mathcal{W}_{1,2,\text{strict}}) = \Omega(p_1 + p_2)$. Computing directly the probability of ties in the multinomial is hard, but we resolve this issue mapping injectively events in which opinion 1 wins with a tie to an event in which opinion 1 wins without a tie. We show that this mapping preserves probabilities up to small constants, therefore we can conclude that events with ties (where opinion 1 wins) have probability weights that are comparable to events without ties (where opinion 1 wins). See Lemma 13 for more details (and note that no initial bias is required for the result of Lemma 13 to hold).

Finally, in Section 4.3 we show how to combine all the ingredients that we have proved in Sections 4.1 and 4.2 to obtain the round-by-round expected growth of the bias and the final statement on plurality consensus. As for the round-by-round expected growth of the bias, we just use the law of total probability which implies that

$$\begin{aligned} & \Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2) && (6)\\ \geq\ & \Pr(\mathcal{W}_{1,2,\text{strict}}) \left[\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})\right] \\ =\ & \Pr(\mathcal{W}_{1,2,\text{strict}}) \left[\Pr(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}})\right], && (7) \end{aligned}$$

and then use Equation (1). Given the expected growth of the bias, we make use of the Bernstein's inequality (Lemma 24 in Section B) to show its round-by-round increase in high probability, and we need to make sure that all other surrounding conditions of the growth are still satisfied in order to iterate in high probability each round.

3.2 Final discussion and open questions

In this paper we present an analysis of the h-majority dynamics when $hp_1 \geq C \log n$ for a large enough constant C. Interestingly, Theorem 2 answers (partially) a long-standing open question on mathoverflow [30] which basically asks how to estimate $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$ when $h = o(1/\delta^2)$, that is, when $\delta = o(\sqrt{1/h})$. With Lemma 16 we answer to this question under the hypothesis that $hp_1 \geq C \log n$, and we also show that the bias increases w.h.p. each round as long as $C'\sqrt{p_1/n} \leq \delta = o(\sqrt{p_1/\log n})$ for some large enough constant $C' > 0$.

We also remark that $\delta \sim \sqrt{p_1/n}$ is the smallest bias that was ever required in the literature on opinion dynamics (see Section 1.2). Indeed, in the proof of Theorem 17, we show that the standard deviation of the bias at the next round is no smaller than some function $\Theta(\sqrt{p_1/n})$, while its expected growth is at least a multiplicative factor $\sim \sqrt{\log n}$. It remains open to understand whether the bias can be reduced for $h \sim k \log n$, which would imply that the growth of the bias is higher than a multiplicative factor $\sim \sqrt{\log n}$: however, we conjecture that this growth is optimal in the regime $h \sim k \log n$.

The main open question here is whether the lower bound $\Omega(k/h^2)$ given by [5] is tight or not. To understand this, one has to improve the assumption on hp_1 and analyze especially the case where $hp_1 \ll 1$. We argue more in the next subsection.

We remark that our work does not deal with perfectly-balanced configurations: one simply has to show that in short time the system reaches a configuration with the bias specified in the hypotheses of Theorem 2. We leave the analysis of the symmetry-breaking phase for future research.

On the multinomial distribution

Our analysis offers a corollary for the multinomial distribution that is interesting per-se, and is just a reformulation of Lemma 13.

▶ **Corollary 5.** *Consider a multinomial random variable $(X_1, \ldots, X_k)$ where each $X_i \sim \text{Binomial}(h, p_i)$, with $p_1 \geq \ldots \geq p_k$. If $hp_1 \geq C \log n$ for a large enough constant $C > 0$, then $\Pr\big(\cap_{i=2}^{k}\{X_1 > X_i\}\big) \geq cp_1$ for some small enough constant $c > 0$.*

We do not know how the statement of Corollary 5 changes when hp_1 decreases, which remains an open question. This is linked to the generalized birthday paradox, which asks as follows: If $p_1 = \ldots = p_k$, what is the probability that there exists $i \in [k]$ such that $X_i > 1$? In fact, if $hp_1^2 = \Theta(1)$, we are in the standard birthday paradox: with constant probability,

we have that $\max_{i\in[k]}\{X_i\} = 1$, which means that whenever opinion 1 is the maximum, the number of ties is h with constant probability. In contrast, Corollary 5 suggests that when $hp_1 = \Omega(\log n)$ the number of ties that involve opinion 1 when it is the maximum is small.

Since the number of ties among the maxima increases when hp_1 gets smaller, we conjecture that, for $hp_1 \ll 1$, $\Pr(\mathcal{W}_{1,\text{strict}}) = o(p_1)$, implying that our approach, that entirely builds around the fact that $\Pr(\mathcal{W}_{1,\text{strict}}) \sim \Pr(\mathcal{W}_1)$, cannot apply, and new ideas are required.

Another open question is to quantify the number of ties among the maxima, in general, and especially when opinion 1 is in the maxima for a generic configuration where $p_1 \geq \ldots \geq p_k$. This would help bridging the two cases $hp_1 \ll 1$ and $hp_1 \gg 1$.

4 Analysis

In this section we present the proof of Theorem 2, following the overview presented in Section 3. Most of the technical proofs either are deferred to Section A or can be found in the full version [17].

4.1 Conditional analysis of the expected behavior of the process

Due to space limitations, the interested reader can find the proofs in the full version [17].

Lemma 3 shows a good lower bound of $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$ for the h-majority when the number of opinion is 2. In order to use this result for k opinions, our plan is to condition on the event that either opinion 1 or 2 wins, without ties. Let $x = \max_{i\geq 3}\{X_i\}$ the mode of the sample among the opinions $\{3,\ldots,k\}$. This event can be written as $\max\{X_1, X_2\} > x$. In the following lemma, we adapt the result of Lemma 3 to work under the additional conditioning event that opinion 1 or 2 is sampled more than x times.

▶ **Lemma 6.** *For any integer $m > 0$, let $Y_1 \sim \text{Binomial}(m, q)$ and $Y_2 = m - Y_1 \sim \text{Binomial}(m, 1-q)$, with $q > 1/2$. We have, for any $m/2 < x \leq m$,*

$$\Pr(Y_1 > Y_2 \mid \max(Y_1, Y_2) > x) - \Pr(Y_2 > Y_1 \mid \max(Y_1, Y_2) > x) \geq C_1 \cdot \min(\sqrt{m}\cdot(2q-1), 1),$$

for some constant $C_1 > 0$.

The next lemma shows that the expected bias at the next round, i.e. $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$, is equal to $\Pr(X_1 > X_2) - \Pr(X_2 > X_1)$, conditioning on the event either opinion 1 or 2 wins, without ties. The last quantity is easier to estimate, because we don't have to handle possible ties. This is to get a formula that resembles that of Lemma 3.

▶ **Lemma 7.** *Recall the Definition 4 of the event $\mathcal{W}_{1,2,\text{strict}}$. We have*

$$\begin{aligned} &\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}}) \\ =\ &\Pr(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}}) \end{aligned}$$

We remark that the statement of Lemma 7 does not hold if we condition on the event $\mathcal{W}_{1,2}$, which includes possible ties. In fact, the event $\{\mathcal{W}_1, \mathcal{W}_{1,2}\}$ contains sub-events for which opinion 1 ties with many other opinions, and we could not find a simple close expression for the probability of these sub-events.

To use Lemma 6 after conditioning on the fact either opinion 1 or 2 wins, we need to estimate the number of total samples of opinion 1 and 2. Since, at the end, we assume $hp_1 \geq \log n$, we can apply the Chernoff bound (Lemma 22 in Section B) to show that $X_1 + X_2 \geq (p_1 + p_2)/2$ w.h.p., but we must ensure that this remains true under the

aforementioned conditioning. The following lemma shows this is indeed the case. Despite the intuitive nature of the statement (the fact either opinion 1 or 2 wins is positively correlated with the absolute value of X_1 and X_2) we need to use an involved coupling argument to formally prove it.

▶ **Lemma 8.** *It holds that*

$$\Pr\left(X_1 + X_2 \geq h \cdot (p_1 + p_2)/2 \mid \mathcal{W}_{1,2,\text{strict}}\right) \geq \Pr\left(X_1 + X_2 \geq h \cdot (p_1 + p_2)/2\right).$$

4.2 Estimating the probability of the conditional event

In this section our goal is to estimate a lower bound for $\Pr(\mathcal{W}_{1,2,\text{strict}})$. More specifically, we show that $\Pr(\mathcal{W}_{1,2,\text{strict}}) \geq C(p_1 + p_2)$ for some constant $C > 0$. As we already argued after Lemma 7, it is important to exclude ties in the event we condition on.

The next definitions introduce terms to refer to opinions that will disappear w.h.p. at the next round, the *weak opinions*, and their counter part, the *strong opinions*.

▶ **Definition 9** (Rare opinions). *Denote with $\mathcal{R}_x := \{i \in [k] : p_i \leq x \cdot p_1\}$ be the set of the x-rare opinions, for $0 < x < 1$.*

▶ **Definition 10** (Strong opinions). *Denote with $\mathcal{S} := \left\{i \in [k] : p_i > \frac{1}{2}p_1\right\}$ be the set of the strong opinions, for $0 < x < 1$.*

Note that the set of the strong opinions and the set of 1/2-rare opinions are complementary.

In the next lemma we show that w.h.p. all weak opinions will be sampled less than the opinion 1, and therefore, they will disappear at the next round. The proof is just an application of Chernoff bounds (Lemma 22 in Section B) and we defer it to Section A.1.

▶ **Lemma 11.** *Let C_2, C_3 be two constants s.t. $0 < C_2 < 1$ and $C_3 > 2$. If $hp_1 > C_4 \log n$, with $C_4 = \left(3\, C_3 \frac{1}{1-C_2}\right)^2$, then*

$$\Pr\left(\cap_{i \in \mathcal{R}_{C_2}} \{X_1 > X_i\}\right) \geq 1 - \frac{1}{n^{C_3 - 2}}.$$

To bound $\Pr(\mathcal{W}_{1,\text{strict}})$, we plan to first show that there exists a constant C s.t. $\mathcal{W}_{1,\text{ties}} \geq Cp_1$, and, therefore, show that $\Pr(\mathcal{W}_{1,\text{strict}}) \geq C \Pr(\mathcal{W}_{1,\text{ties}})$, for some other constant C. Hence, in the next lemma we show that $\Pr(\mathcal{W}_1) \geq Cp_1$ which, in turn, implies that $\Pr(\mathcal{W}_{1,\text{ties}}) \geq Cp_1$. The idea is to use Lemma 11 to show that only strong opinion compete for the win and in the worst case they have all same probability to win without ties. The thesis follows after showing that the number of strong opinions is at most $2/p_1$, and that $\sum_{i \in \mathcal{S}} \Pr(\mathcal{W}_i) \geq 1$. Again, the proof is deferred to Section A.1.

▶ **Lemma 12.** *Let $C_4 = (18)^2$. If $p_1 h \geq C_4 \log n$, then*

$$\Pr(\mathcal{W}_1) \geq \frac{1}{3} p_1.$$

The next Lemma plays a key role in avoiding the computation of the probability of ties. Assuming that $hp_1 = O(\log n)$, we can map each realization of the multinomial in which opinion 1 wins with ties to a realization in which opinion 1 wins without ties. We will show that these two realizations have the same probability up to a constant. Therefore, we can conclude that tie events have a comparable weight to events without ties. We present its proof in Section A.1 as we believe this is an interesting result per-se, which also proves Corollary 5 in Section 3.2.

▶ **Lemma 13.** *Let $hp_1 > C_4 \log n$ for $C_4 = 18^2$. It holds that*

$$\Pr(\mathcal{W}_{1,strict}) \geq \frac{1}{6}\Pr(\mathcal{W}_{1,\text{ties}}) \geq \frac{1}{18}p_1.$$

The goal of this section was to show that $\Pr(\mathcal{W}_{1,2,\text{strict}}) \geq C(p_1+p_2)$ for some constant $C > 0$. This can be easily obtained by applying Lemma 13, because $p_1 + p_2 \leq 2p_1$ and $\mathcal{W}_{1,\text{strict}} \subseteq \mathcal{W}_{1,2,\text{strict}}$.

▶ **Corollary 14.** *Let $hp_1 > C_4 \log n$ for $C_4 = 18^2$. It holds that*

$$\Pr(\mathcal{W}_{1,2,strict}) \geq \frac{1}{36}(p_1+p_2).$$

4.3 Putting everything together

In this section we can finally put together all the results to give a lower bound to $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_j)$ and to use it to prove Theorem 2. Next lemma is the statement that is "equivalent in spirit" to Lemma 3 but for many opinions, conditional on $\mathcal{W}_{1,2,\text{strict}}$. Its proof involves a mix of the previous results and is quite intricate.

▶ **Lemma 15.** *Let $p_1\, h \geq C_4 \log n$, with $C_4 = 18^2$. We have*

$$\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,strict}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,strict}) \geq C \min\left\{\frac{(p_1-p_2)\sqrt{h}}{\sqrt{2(p_1+p_2)}}, 1\right\},$$

for some constant $C > 0$.

Proof of Lemma 15. Let

$$M = \{m \in \mathbb{N} : \Pr(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}) > 0\} \cap \{m \geq h \cdot (p_1+p_2)/2\}$$

and

$$L_m = \{x \in \mathbb{N} : \Pr\left(\max_{j\geq 3} X_j = x \mid X_1 + X_2 = m, \mathcal{W}_{1,2,\text{strict}}\right) > 0\}.$$

We have

$$\begin{aligned}
&\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}}) \\
\underset{(i)}{=}\ &\Pr(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}}) \\
\underset{(ii)}{\geq}\ &\sum_{m\in M} \Pr(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}) \left[\Pr(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}, X_1 + X_2 = m)\right. \\
&\left. - \Pr(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}}, X_1 + X_2 = m)\right] \\
\underset{(iii)}{=}\ &\sum_{m\in M} \Pr(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}) \\
&\cdot \sum_{x\in L_m} \Pr\left(\max_{j\geq 3} X_j = x \mid X_1 + X_2 = m, \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot \left[\Pr\left(X_1 > X_2 \mid \mathcal{W}_{1,2,\text{strict}}, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right)\right. \\
&\left. - \Pr\left(X_2 > X_1 \mid \mathcal{W}_{1,2,\text{strict}}, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right)\right] \\
\underset{(iv)}{=}\ &\sum_{m\in M} \Pr(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}})
\end{aligned}$$

$$
\begin{aligned}
&\cdot \sum_{x \in L_m} \Pr\left(\max_{j\geq 3} X_j = x \mid X_1 + X_2 = m, \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot \left[\Pr\left(X_1 > X_2 \mid \max\{X_1, X_2\} > x, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right)\right. \\
&\left. - \Pr\left(X_2 > X_1 \mid \max\{X_1, X_2\} > x, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right)\right], \qquad (8)
\end{aligned}
$$

where in (i) we used Lemma 7, in $(ii), (iii)$ we used the law of total probabilities, and in (iv) we used that, by Definition 4,

$$
\begin{aligned}
&\left(\mathcal{W}_{1,2,\text{strict}}, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right) \\
= &\left(\max\{X_1, X_2\} > x, X_1 + X_2 = m, \max_{j\geq 3} X_j = x\right).
\end{aligned}
$$

Conditioning on $\{X_1 + X_2 = m\}$, makes X_1, X_2 independent from $X_3, \ldots, X_k$, and distributed as Y_1, Y_2, where $Y_1 \sim \text{Binomial}(m, q_1)$ with $q_1 = \frac{p_1}{p_1+p_2}$ and $Y_2 = m - Y_1$. We obtain, by Equation (8)

$$
\begin{aligned}
&\Pr\left(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}\right) - \Pr\left(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}}\right) \\
\geq &\sum_{m \in M} \Pr\left(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot \sum_{x \in L_m} \Pr\left(\max_{j\geq 3} X_j = x \mid X_1 + X_2 = m, \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot \left[\Pr\left(Y_1 > Y_2 \mid \max\{Y_1, Y_2\} \geq x\right) - \Pr\left(Y_2 > Y_1 \mid \max\{Y_1, Y_2\} \geq x\right)\right] \\
\underset{(i)}{\geq} &\sum_{m \in M} \Pr\left(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot \sum_{x \in L_m} \Pr\left(\max_{j\geq 3} X_j = x \mid X_1 + X_2 = m, \mathcal{W}_{1,2,\text{strict}}\right) \\
&\cdot C_1 \cdot \min\left(\sqrt{m}\left(\frac{\delta_2}{p_1+p_2}\right), 1\right) \\
\underset{(ii)}{=} &\sum_{m \in M} \Pr\left(X_1 + X_2 = m \mid \mathcal{W}_{1,2,\text{strict}}\right) \cdot C_1 \cdot \min\left(\sqrt{m}\left(\frac{\delta_2}{p_1+p_2}\right), 1\right) \\
\underset{(iii)}{\geq} &\Pr\left(X_1 + X_2 \geq h \cdot (p_1+p_2)/2 \mid \mathcal{W}_{1,2,\text{strict}}\right) \cdot C_1 \cdot \min\left(\left(\frac{\delta_2\sqrt{h}}{\sqrt{2(p_1+p_2)}}\right), 1\right) \\
\underset{(iv)}{\geq} &C_1\left(1 - \exp\left(-\frac{1}{8}h(p_1+p_2)\right)\right) \cdot \min\left(\left(\frac{\delta_2\sqrt{h}}{\sqrt{2(p_1+p_2)}}\right), 1\right) \\
\underset{(v)}{\geq} &C_1\left(1 - \frac{1}{n^2}\right) \cdot \min\left(\left(\frac{\delta_2\sqrt{h}}{\sqrt{2(p_1+p_2)}}\right), 1\right) \\
\geq &C \min\left(\left(\frac{\delta_2\sqrt{h}}{\sqrt{2(p_1+p_2)}}\right), 1\right),
\end{aligned}
$$

where in (i) we used Lemma 6, in (ii) we used the law of total probabilities, in (iii) we used that by definition of M, $m \geq h(p_1+p_2)/2$, in (iv) we used Lemma 8 and the multiplicative Chernoff bound (Lemma 22 in Section B), and in (v) we used that $hp_1 \geq 16 \log n$. This concludes the proof of Lemma 15. ◀

In the next lemma, we finally remove the conditional event $\mathcal{W}_{1,2,\text{strict}}$ and bound $\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2)$ from below. We need to use Lemmas 13 and 15, but its proof consists in just mixing in the "right way" previous results, hence we defer it to Section A.2. From now on, we denote the difference $p_1 - p_j$ by $\delta(j)$.

▶ **Lemma 16.** *Let $p_1\, h \geq C_4 \log n$, with $C_4 = 18^2$. If $\delta(j) \leq \sqrt{\frac{2(p_1+p_2)}{h}}$, then there exists a constant $C_5 > 0$ s.t.*

$$\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_j) \geq C_5 \left(\delta(j)\sqrt{\frac{h}{p_1+p_2}}\right) \Pr(\mathcal{W}_1).$$

Instead, if $\delta(j) \geq \sqrt{\frac{2(p_1+p_2)}{h}}$, there exists a constant $0 < C_6 < 1$ s.t.

$$\Pr(\mathcal{W}_1) \geq \frac{1}{1-C_6} \Pr(\mathcal{W}_j).$$

Given a configuration $(p_1, \ldots, p_k)$, we denote by p'_j the probability associated to opinion j at the next round. Fix an opinion $j \neq 1$. Let $\delta(j)'$ be the bias between opinion 1 and opinion j at the next round, i.e. $p'_1 - p'_j$.

Before proceeding in the analysis presentation, let us comment Lemma 16. Note that $\Pr(\mathcal{W}_j)$ is the expected value of p'_j. In the case the bias between opinion 1 and opinion j is more than $\sqrt{\frac{2(p_1+p_2)}{h}}$, at the next round we have $p'_1 \geq (1+C)p'_j$ in expectation, for some constant $C > 0$. In the case the bias is smaller than $\sqrt{\frac{2(p_1+p_2)}{h}}$, by the fact $h = \Omega(p_1 \log n)$ and $\Pr(\mathcal{W}_1) = \Omega(p_1 + p_2)$, Lemma 16 implies that at the next round the expected bias grows of a $\Omega(\log n)$ factor. Thanks to standard concentration bounds (Chernoff Bound and Bernstein Inequality) we show that w.h.p. the new bias is sufficiently close to its expectation.

Now we all have all ingredients to prove Theorem 2. Next theorem is just a reformulation of Theorem 2.

▶ **Theorem 17.** *Consider an initial configuration $(p_1, \ldots, p_k)$ s.t. $p_1 \geq C_7 \frac{\log n}{n}$, and $p_1 - p_j \geq C_8 \sqrt{\frac{p_1}{n}}$ for all $j \geq 2$. Then the h-majority dynamics with $hp_1 > C_4 \log n$ converges in time $O(\log n)$ to opinion 1, w.h.p., for some constants $C_4, C_7, C_8 > 0$.*

The proof relies on first quantifying the increase in the bias between the first opinion and other opinions after one round. Then it iterates the same reasoning until consensus (for the complete proof, see the full version [17]). For the first point, we use concentration bounds on the results obtained in Lemma 16. In particular, we distinguish three sub-cases based on the size of the bias between two opinions. For the second point, we need to show that the initial hypotheses are still satisfied after one round to ensure that we can iterate.

In the next lemma, we show that if $p_1 \geq (1+C)p_j$ for some constant $C > 0$, opinion j disappears at the next round, by simply applying the Chernoff bound.

▶ **Lemma 18** (Opinion j-disappear at the next round)**.** *Let $hp_1 \geq C_4 \log n$. If $\delta(j) \geq \left(1 - \frac{1}{1+C_6}\right) p_1$ for the constant C_6 defined in Lemma 16, with probability at least $1 - \frac{1}{n^4}$ opinion j will disappear at the next round*

The next Lemma shows that if the bias is $\Omega\left(\sqrt{p_1/h}\right)$, then the new bias will fall within the hypothesis of Lemma 18 in one step. Therefore, we conclude that, in two steps, opinion j will disappear if the bias is large enough. The proof relies on the results obtained in Lemma 16, combined with the Chernoff bound.

▶ **Lemma 19** (Opinion j-disappear in two rounds). *Let $hp_1 \geq C_4 \log n$. If $p_1 \geq C_7 \frac{\log n}{n}$ and $\sqrt{\frac{2(p_1+p_2)}{h}} \leq \delta(j) \leq \left(1 - \frac{1}{1+C_6}\right) p_1$ for the constant $C_6 > 0$ defined in Lemma 16 and for a constant $C_7 > 0$, we have that*

$$\Pr\left(\delta(j)' \geq \left(1 - \frac{1}{1+C_6}\right) p_1'\right) \geq 1 - \frac{2}{n^4}.$$

The next Lemma addresses the minimum possible bias. We show that, until we fall back into the hypotheses of Lemmas 18 and 19, the bias grows exponentially. The proof is more complex than the previous two lemmas because it requires bounding the second moment of the bias to apply the Bernstein inequality.

▶ **Lemma 20** (Bias exponential growth). *Let $hp_1 \geq C_4 \log n$. If $p_1 \geq C_7 \frac{\log n}{n}$ and $C_8 \sqrt{\frac{p_1}{n}} \leq \delta(j) \leq \sqrt{\frac{2(p_1+p_2)}{h}}$ for the constant $C_6 > 0$ defined in Lemma 16 and for some constants $C_7, C_8 > 0$, we have that*

$$\Pr\left(\delta(j)' \geq e\,\delta(j)\right) \geq 1 - \frac{1}{n^4}$$

The next Lemma ensures that the hypothesis of Theorem 2 on the bias is always satisfied so that we can iterate Lemma 20.

▶ **Lemma 21** (The new bias satisfies the initial hypothesis). *Let $\delta(j) \geq \sqrt{\frac{p_1}{n}}$, and a constant $C_8 > 0$. For n large enough we have*

$$\Pr\left(\delta(j)' \geq C\sqrt{\frac{p_1'}{n}}\right) \geq 1 - \frac{2}{n^4}.$$

References

1 Talley Amir, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. Fast convergence of k-opinion undecided state dynamics in the population protocol model. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 13–23. ACM, 2023. doi:10.1145/3583668.3594589.

2 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Comput.*, 21(2):87–102, 2008. doi:10.1007/S00446-008-0059-Z.

3 Gregor Bankhamer, Petra Berenbrink, Felix Biermeier, Robert Elsässer, Hamed Hosseinpour, Dominik Kaaser, and Peter Kling. Fast consensus via the unconstrained undecided state dynamics. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3417–3429. SIAM, 2022. doi:10.1137/1.9781611977073.135.

4 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. Plurality consensus in the gossip model. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 371–390. SIAM, 2015. doi:10.1137/1.9781611973730.27.

5 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, Riccardo Silvestri, and Luca Trevisan. Simple dynamics for plurality consensus. *Distributed Computing*, 30(4):293–306, August 2017. doi:10.1007/s00446-016-0289-4.

6 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 620–635. SIAM, 2016. doi:10.1137/1.9781611974331.CH46.

7 Luca Becchetti, Andrea Clementi, Francesco Pasquale, Luca Trevisan, Robin Vacus, and Isabella Ziccardi. The minority dynamics and the power of synchronicity. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4155–4176. SIAM, 2024. doi:10.1137/1.9781611977912.144.

8 Petra Berenbrink, Felix Biermeier, and Christopher Hahn. Undecided state dynamics with stubborn agents. *CoRR*, abs/2406.07335, 2024. doi:10.48550/arXiv.2406.07335.

9 Petra Berenbrink, Andrea Clementi, Robert Elsässer, Peter Kling, Frederik Mallmann-Trenn, and Emanuele Natale. Ignore or comply?: On breaking symmetry in consensus. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 335–344. ACM, 2017. doi:10.1145/3087801.3087817.

10 Petra Berenbrink, Amin Coja-Oghlan, Oliver Gebhard, Max Hahn-Klimroth, Dominik Kaaser, and Malin Rau. On the hierarchy of distributed majority protocols. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, volume 253 of *LIPIcs*, pages 23:1–23:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.OPODIS.2022.23.

11 Andrea Clementi, Francesco D'Amore, George Giakkoupis, and Emanuele Natale. Search via Parallel Lévy Walks on Z2. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 81–91. ACM, 2021. doi:10.1145/3465084.3467921.

12 Andrea Clementi, Mohsen Ghaffari, Luciano Gualà, Emanuele Natale, Francesco Pasquale, and Giacomo Scornavacca. A tight analysis of the parallel undecided-state dynamics with two colors. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.MFCS.2018.28.

13 Colin Cooper, Frederik Mallmann-Trenn, Tomasz Radzik, Nobutaka Shimizu, and Takeharu Shiraga. Asynchronous 3-majority dynamics with many opinions. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4095–4131. SIAM, 2025. doi:10.1137/1.9781611978322.140.

14 Emilio Cruciani, Hlafo Alfie Mimun, Matteo Quattropani, and Sara Rizzo. Phase transition of the k-majority dynamics in biased communication models. *Distributed Comput.*, 36(2):107–135, 2023. doi:10.1007/S00446-023-00444-2.

15 Francesco D'Amore, Andrea Clementi, and Emanuele Natale. Phase transition of a non-linear opinion dynamics with noisy interactions - (extended abstract). In Andréa Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity - 27th International Colloquium, SIROCCO 2020, Paderborn, Germany, June 29 - July 1, 2020, Proceedings*, volume 12156 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2020. doi:10.1007/978-3-030-54921-3_15.

16 Francesco D'Amore, Andrea Clementi, and Emanuele Natale. Phase transition of a nonlinear opinion dynamics with noisy interactions. *Swarm Intell.*, 16(4):261–304, 2022. doi:10.1007/S11721-022-00217-W.

17 Francesco d'Amore, Niccolò D'Archivio, George Giakkoupis, and Emanuele Natale. On the h-majority dynamics with many opinions, 2025. doi:10.48550/arXiv.2506.20218.

18 Francesco D'Amore and Isabella Ziccardi. Phase transition of the 3-majority dynamics with uniform communication noise. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2022. doi:10.1007/978-3-031-09993-9_6.

19 Francesco D'Amore and Isabella Ziccardi. Phase transition of the 3-majority opinion dynamics with noisy interactions. *Theor. Comput. Sci.*, 1028:115030, 2025. doi:10.1016/J.TCS.2024.115030.

20 Niccolò D'Archivio and Robin Vacus. On the limits of information spread by memory-less agents. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DISC.2024.18.

21 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

22 Ofer Feinerman, Bernhard Haeupler, and Amos Korman. Breathe before speaking: efficient information dissemination despite noisy, limited and anonymous communication. In Magnús M. Halldórsson and Shlomi Dolev, editors, *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 114–123. ACM, 2014. doi:10.1145/2611462.2611469.

23 Ofer Feinerman and Amos Korman. The ANTS problem. *Distributed Comput.*, 30(3):149–168, 2017. doi:10.1007/S00446-016-0285-8.

24 Pierre Fraigniaud, Amos Korman, and Yoav Rodeh. Parallel exhaustive search without coordination. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 312–323. ACM, 2016. doi:10.1145/2897518.2897541.

25 Pierre Fraigniaud and Emanuele Natale. Noisy rumor spreading and plurality consensus. *Distributed Computing*, 32(4):257–276, 2019. doi:10.1007/S00446-018-0335-5.

26 Matthias Függer, Thomas Nowak, and Joel Rybicki. Majority consensus thresholds in competitive lotka-volterra populations. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 76–86. ACM, 2024. doi:10.1145/3662158.3662823.

27 Mohsen Ghaffari and Johannes Lengler. Nearly-tight analysis for 2-choice and 3-majority consensus dynamics. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 305–313. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212738.

28 George Giakkoupis, Volker Turau, and Isabella Ziccardi. Self-stabilizing MIS computation in the beeping model. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain*, volume 319 of *LIPIcs*, pages 28:1–28:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DISC.2024.28.

29 George Giakkoupis and Isabella Ziccardi. Distributed self-stabilizing MIS with few states and weak communication. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 310–320. ACM, 2023. doi:10.1145/3583668.3594581.

30 Immanuel Weihnachten (https://mathoverflow.net/users/23351/immanuel weihnachten). Lower bound on the probability of guessing the mode in a small multinomial sample. MathOverflow. URL:https://mathoverflow.net/q/210018 (version: 2015-06-24). arXiv:https://mathoverflow.net/q/210018.

31 Amos Korman and Robin Vacus. Distributed alignment processes with samples of group average. *IEEE Trans. Control. Netw. Syst.*, 10(2):960–971, 2023. doi:10.1109/TCNS.2022.3212640.

32 Hicham Lesfari, Frédéric Giroire, and Stéphane Pérennes. Biased majority opinion dynamics: Exploiting graph k-domination. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 377–383. ijcai.org, 2022. doi:10.24963/IJCAI.2022/54.

33 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

34 Nobutaka Shimizu and Takeharu Shiraga. 3-majority and 2-choices with many opinions. *CoRR*, abs/2503.02426, 2025. To appear at PODC 2025. doi:10.48550/arXiv.2503.02426.

35 Robin Vacus and Isabella Ziccardi. Minimalist Leader Election Under Weak Communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '25, pages 406–416, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3732772.3733559.

A Missing proofs

A.1 Omitted proofs from Section 4.2

Proof of Lemma 11. Define the event $\mathcal{C}_j = \left\{|X_j - hp_j| < \sqrt{3C_3\, hp_j \log n}\right\}$ and $\mathcal{C} = \cap_{j=1}^{k} \mathcal{C}_j$. The event $\mathcal{C}_j$ can be rewritten as

$$\mathcal{C}_j = \left\{X_j \in \left(hp_j - \sqrt{3C_3\, hp_j \log n}, hp_j + \sqrt{3C_3\, hp_j \log n}\right)\right\}. \tag{9}$$

For $i \in \mathcal{R}$, we have

$$\begin{aligned} & hp_1 - \sqrt{3\, C_3\, hp_1 \log n} - \left(hp_i + \sqrt{3\, C_3\, hp_i \log n}\right) \\ \geq\ & (1 - C_2)\, hp_1 - \sqrt{3\, C_3\, hp_1 \log n} && (i \in \mathcal{R} \implies hp_i \leq hC_2 \cdot p_1) \\ >\ & 0, \end{aligned}$$

by taking $hp_1 > C_4 \log n$, with $C_4 = \left(3\, C_3 \frac{1}{1-C_2}\right)^2$. This fact, together with Equation (9), implies that $\mathcal{C} \subset \left\{\cap_{i \in \mathcal{R}_{C_2}} \{X_1 > X_i\}\right\}$, and therefore $\Pr\left(\cap_{i \in \mathcal{R}_{C_2}} \{X_1 > X_i\}\right) \geq \Pr(\mathcal{C})$. By the multiplicative Chernoff bound we have

$$\begin{aligned} \Pr(C_j) &= \Pr\left(|X_j - hp_j| < \sqrt{3C_3\, hp_j \log n}\right) \\ &= 1 - \Pr\left(|X_j - hp_j| \geq \sqrt{3C_3\, hp_j \log n}\right) \\ &\geq 1 - 2\exp\left(-\frac{3C_3 \log n}{3}\right) \\ &= 1 - \frac{2}{n^{C_3}}, \end{aligned}$$

and by the union bound and by the fact the number of opinion $k \leq n$, we obtain

$$\Pr\left(\cap_{i \in \mathcal{R}} \{X_1 > X_i\}\right) \geq \Pr(\mathcal{C}) \geq 1 - \frac{2k}{n^{C_3}} \geq 1 - \frac{1}{n^{C_3 - 2}},$$

concluding the proof of Lemma 11. ◀

Proof of Lemma 12. Recall $\mathcal{S} := \left\{i \in [k] : p_i > \frac{1}{2} p_1\right\}$ be the set of the *strong opinions*. By definition of $\mathcal{S}$ we have

$$1 \geq \sum_{j \in \mathcal{S}} p_j > \frac{|\mathcal{S}|\, p_1}{2},$$

and therefore we have

$$|\mathcal{S}| \leq \frac{2}{p_1}. \tag{10}$$

Recall that $\mathcal{R}_{1/2} := \left\{i \in [k] : p_i \leq \frac{1}{2}p_1\right\}$ is the set of the 1/2-*rare opinions*. If $i \in \mathcal{R}_{1/2}$, we have that

$$\Pr\left(\mathcal{W}_i \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) = 0.$$

By this fact, and by noticing that an opinion is either strong or 1/2-rare,

$$1 = \Pr\left(\bigcup_{j \in \mathcal{S}} \mathcal{W}_j \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \leq \sum_{j \in \mathcal{S}} \Pr\left(\mathcal{W}_j \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right). \tag{11}$$

Since $X_j \sim \text{Binomial}(p_j, h)$ and $p_1 \geq p_j$, for all $j \in [k]$, we have that X_1 stochastically dominates X_j for all $j \geq 2$. This fact remains true also conditioning on the event $\left\{\cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right\}$. Therefore, we have

$$\Pr\left(\mathcal{W}_1 \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \geq \Pr\left(\mathcal{W}_j \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right),$$

for all $j \in \mathcal{S}, j \geq 2$. This fact, together with Equation (11), implies

$$|\mathcal{S}| \cdot \Pr\left(\mathcal{W}_1 \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \geq \sum_{j \in \mathcal{S}} \Pr\left(\mathcal{W}_j \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \geq 1,$$

and, by using Equation (10), we obtain

$$\Pr\left(\mathcal{W}_1 \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \geq \frac{1}{|\mathcal{S}|} \geq \frac{p_1}{2}.$$

We can conclude the proof of Lemma 12 noticing that, by Lemma 11 with $C_3 = 3$ and $C_2 = 1/2$ (which implies $C_4 = (18)^2$, we have

$$\Pr(\mathcal{W}_1) \geq \Pr\left(\cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \cdot \Pr\left(\mathcal{W}_1 \mid \cap_{\ell \in \mathcal{R}_{1/2}} \{X_1 > X_\ell\}\right) \geq \left(1 - \frac{1}{n}\right)\frac{p_1}{2} \geq \frac{p_1}{3},$$

for n large enough. ◀

Proof of Lemma 13. Define the event $\mathcal{C}_j = \left\{|X_j - hp_j| < \sqrt{9hp_j \log n}\right\}$ and $\mathcal{C} = \cap_{j \geq 1}\mathcal{C}_j$. By the multiplicative Chernoff bound (Lemma 22 in Section B), we obtain

$$\begin{aligned}
\Pr(\mathcal{C}_j) &= \Pr\left(|X_j - hp_j| < \sqrt{9\,hp_j \log n}\right) \\
&= 1 - \Pr\left(|X_j - hp_j| \geq \sqrt{9\,hp_j \log n}\right) \\
&\geq 1 - 2\exp\left(-\frac{9\log n}{3}\right) \\
&= 1 - \frac{2}{n^3}.
\end{aligned}$$

Moreover, by the union bound and by the fact $k \leq n$, we have, for n large enough,

$$\Pr(\mathcal{C}) \geq 1 - \frac{2k}{n^3} \geq 1 - \frac{1}{n}. \tag{12}$$

Recall $\mathcal{S} := \left\{i \in [k] : p_i > \frac{1}{2}p_1\right\}$ be the set of the *strong opinions*. We first note that

$$\begin{aligned}
\mathcal{C} &\subseteq \bigcap_{j\in\mathcal{S}} \left\{X_j > hp_j - \sqrt{9hp_j \log n}\right\} \\
&\subseteq \bigcap_{j\in\mathcal{S}} \left\{X_j > \frac{1}{2}hp_1 - \sqrt{9hp_1 \log n}\right\} \qquad \left(p_i > \frac{1}{2}p_1, p_i \le p_1, \text{ for all } i \in \mathcal{S}\right) \\
&\subseteq \cap_{j\in\mathcal{S}} \{X_j > 0\}. \qquad (hp_1 > 36 \log n) \qquad (13)
\end{aligned}$$

Let us denote a particular realization of a multinomial in the following way.

$$\mathcal{X}(x_1, \ldots, x_k) = \{X_1 = x_1, \ldots, X_k = x_k\}.$$

We define the event $\mathcal{X}(x_1, \ldots, x_k)$ a **1-tie** if $x_1 \ge x_i$ for all $i \ge 1$ and there exists a $j \ne 1$ s.t. $x_j = x_1$. In other words, opinion 1 is the most sampled, together with at least another opinion. We denote the set of 1-tie as $\mathcal{T}_1$.

We map a 1-tie to an event in which the opinion 1 gets one extra sample which is stolen from the least scoring strong opinion. Formally, if $\mathcal{X}(x_1, \ldots, x_k) \in \mathcal{T}_1$ and $j = \max\{i : X_i = \min_{r\in\mathcal{S}}\{X_r\}\}$, we define:

$$f : \mathcal{X}(x_1, \ldots, x_k) \mapsto \mathcal{X}(x_1 + 1, \ldots, x_{j-1}, x_j - 1, x_{j+1}, \ldots, x_k).$$

The map is well-defined if and only if $x_j > 0$. If $\mathcal{X}(x_1, \ldots, x_k) \in \mathcal{C}$, by Equation (13), we have $\mathcal{X}(x_1, \ldots, x_k) \in \cap_{j\in\mathcal{S}}\{X_j > 0\}$, and, since $j \in \mathcal{S}$ by definition, we obtain that $x_j > 0$. Therefore, f is well-defined on $\mathcal{C}$.

Moreover, the map is injective. Using the probability mass function of the multinomial distribution, we compute the following:

$$\frac{\Pr\left(f\left(\mathcal{X}(x_1, \ldots, x_k)\right)\right)}{\Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right)} = \frac{x_j}{x_1 + 1} \cdot \frac{p_1}{p_j} \ge \frac{x_j}{x_1 + 1},$$

where we used that $p_1 \ge p_j$. Therefore we obtain

$$\begin{aligned}
&\sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \Pr\left(f\left(\mathcal{X}(x_1, \ldots, x_k)\right)\right) \\
&\ge \sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \frac{x_j}{x_1 + 1} \Pr\left(\left(\mathcal{X}(x_1, \ldots, x_k)\right)\right) \\
&\ge \frac{hp_j - \sqrt{9hp_j \log n}}{1 + hp_1 + \sqrt{9hp_1 \log n}} \cdot \sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right) \qquad \text{(by definition of } \mathcal{C}) \\
&\ge \frac{\frac{1}{2}hp_1 - \sqrt{9hp_1 \log n}}{1 + hp_1 + \sqrt{9hp_1 \log n}} \cdot \sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right) \quad \left(p_i > \frac{1}{2}p_1, p_i \le p_1, \text{ for } i \in \mathcal{S}\right) \\
&= \frac{\frac{1}{2}\sqrt{\frac{hp_1}{9\log n}} - 1}{\sqrt{\frac{hp_1}{9\log n}} + 2} \cdot \sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right) \\
&\ge \frac{1}{4} \cdot \sum_{\mathcal{X}\in\mathcal{T}_1, \mathcal{X}\in\mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right), \qquad (14)
\end{aligned}$$

where the last inequality holds because $x \mapsto \frac{\frac{1}{2}x-1}{x+2}$ is increasing and $hp_1 \ge 18^2 \log n$. Recall Definition 4. Since $\mathcal{T}_1 = \mathcal{W}_{1,\text{ties}} \setminus \mathcal{W}_{1,\text{strict}}$ and $f(\mathcal{T}_1 \cap \mathcal{C}) \subseteq \mathcal{W}_1$, we can write

$$\Pr\left(\mathcal{W}_{1,\text{ties}}, \mathcal{C}\right)$$

$$
\begin{aligned}
&= \Pr\left(\mathcal{W}_{1,\text{strict}}, \mathcal{C}\right) + \Pr\left(\mathcal{W}_{1,\text{ties}} \setminus \mathcal{W}_{1,\text{strict}}, \mathcal{C}\right) \\
&\geq \sum_{\mathcal{X} \in \mathcal{T}_1, \mathcal{X} \in \mathcal{C}} \Pr\left(f\left(\mathcal{X}(x_1, \ldots, x_k)\right)\right) + \sum_{\mathcal{X} \in \mathcal{T}_1, \mathcal{X} \in \mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right) \\
&\geq \left(1 + \frac{1}{4}\right) \sum_{\mathcal{X} \in \mathcal{T}_1, \mathcal{X} \in \mathcal{C}} \Pr\left(\mathcal{X}(x_1, \ldots, x_k)\right) && \text{(by Equation (14))} \\
&= \left(1 + \frac{1}{4}\right) \Pr\left(\mathcal{W}_{1,\text{ties}} \setminus \mathcal{W}_{1,\text{strict}}, \mathcal{C}\right).
\end{aligned}
$$

Hence,

$$
\Pr\left(\mathcal{W}_{1,\text{ties}} \setminus \mathcal{W}_{1,\text{strict}}, \mathcal{C}\right) \leq \frac{4}{5} \cdot \Pr\left(\mathcal{W}_{1,\text{ties}}, \mathcal{C}\right),
$$

and

$$
\begin{aligned}
&\Pr\left(\mathcal{W}_{1,\text{strict}}\right) \geq \Pr\left(\mathcal{W}_{1,\text{strict}}, \mathcal{C}\right) = \Pr\left(\mathcal{W}_{1,\text{ties}}, \mathcal{C}\right) - \Pr\left(\mathcal{W}_{1,\text{ties}} \setminus \mathcal{W}_{1,\text{strict}}, \mathcal{C}\right) \\
&\geq \frac{1}{5} \cdot \Pr\left(\mathcal{W}_{1,\text{ties}}, \mathcal{C}\right) \\
&= \frac{1}{5}\left(\mathcal{W}_{1,\text{ties}} + \Pr\left(\mathcal{C}\right) - \Pr\left(\mathcal{W}_{1,\text{ties}} \cup \mathcal{C}\right)\right) && \text{(by the inclusion–exclusion princ.)} \\
&\geq \frac{1}{5}\left(\mathcal{W}_{1,\text{ties}} + \Pr\left(\mathcal{C}\right) - 1\right) && \left(\Pr\left(\mathcal{W}_{1,\text{ties}} \cup \mathcal{C}\right) \leq 1\right) \\
&\geq \frac{1}{5}\left(\mathcal{W}_{1,\text{ties}} - \frac{1}{n}\right) && \text{(by Equation (12))} \\
&\geq \frac{1}{6}\mathcal{W}_{1,\text{ties}} \geq \frac{1}{18} p_1
\end{aligned}
$$

for n large enough. This concludes the proof of Lemma 13. ◀

Proof of Corollary 14. By Lemmas 12 and 13, we obtain

$$
\Pr\left(\mathcal{W}_{1,2,\text{strict}}\right) \geq \Pr\left(\mathcal{W}_{1,\text{strict}}\right) \geq \frac{1}{6}\Pr(\mathcal{W}_{1,\text{ties}}) \geq \frac{1}{18}\, p_1 = \frac{1}{36}\left(p_1 + p_1\right) \geq \frac{1}{36}(p_1 + p_2).
$$

This concludes the proof of Corollary 14. ◀

A.2 Omitted proofs from Section 4.3

Proof of Lemma 16. Since $p_2 \geq p_j$ for all $j \geq 2$, we have that $\Pr\left(\mathcal{W}_j\right) \leq \Pr\left(\mathcal{W}_2\right)$, and, therefore,

$$
\Pr\left(\mathcal{W}_1\right) - \Pr\left(\mathcal{W}_2\right) \implies \Pr\left(\mathcal{W}_1\right) - \Pr\left(\mathcal{W}_j\right),
$$

and,

$$
\Pr\left(\mathcal{W}_1\right) \geq \frac{1}{1 - C_6} \Pr\left(\mathcal{W}_2\right) \implies \Pr\left(\mathcal{W}_1\right) \geq \frac{1}{1 - C_6} \Pr\left(\mathcal{W}_j\right).
$$

Hence, we will focus on the proof statement for $j = 2$.

Consider the case $\delta(2) \leq \sqrt{\frac{2(p_1+p_2)}{h}}$, that, in particular, implies that

$$
\min\left(\left(\frac{\delta(2)\sqrt{h}}{\sqrt{2(p_1 + p_2)}}\right), 1\right) = \left(\frac{\delta(2)\sqrt{h}}{\sqrt{2(p_1 + p_2)}}\right).
$$

We have

$$\begin{aligned}
&\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2) \\
&\geq \Pr(\mathcal{W}_{1,2,\text{strict}}) \cdot [\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})] \\
&\geq \Pr(\mathcal{W}_{1,\text{strict}}) \cdot [\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})] && (\mathcal{W}_{1,\text{strict}} \subseteq \mathcal{W}_{1,2,\text{strict}}) \\
&\geq \frac{1}{6}\Pr(\mathcal{W}_{1,\text{ties}}) \cdot [\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})] && (\text{by Lemma 13}) \\
&\geq \frac{1}{6}\Pr(\mathcal{W}_1) \cdot [\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})] && (\mathcal{W}_1 \subseteq \mathcal{W}_{1,\text{ties}}) \\
&\geq \frac{C}{6}\Pr(\mathcal{W}_1)\,\delta(2)\sqrt{\frac{h}{p_1+p_2}} && (\text{by Lemma 15})
\end{aligned}$$

Setting $C_5 = \frac{C}{6}$, we conclude the proof of the first statement of Lemma 16. Now consider the case $\delta \geq \sqrt{\frac{2(p_1+p_2)}{h}}$, that implies that $\left(\frac{\delta\sqrt{h}}{\sqrt{2(p_1+p_2)}}\right) = 1$.

$$\begin{aligned}
&\Pr(\mathcal{W}_1) - \Pr(\mathcal{W}_2) \\
&\geq \Pr(\mathcal{W}_{1,2,\text{strict}}) \cdot [\Pr(\mathcal{W}_1 \mid \mathcal{W}_{1,2,\text{strict}}) - \Pr(\mathcal{W}_2 \mid \mathcal{W}_{1,2,\text{strict}})] \\
&\geq \Pr(\mathcal{W}_{1,2,\text{strict}}) \cdot C && (\text{by Lemma 15}) \\
&\geq \Pr(\mathcal{W}_{1,\text{strict}}) \cdot C && (\mathcal{W}_{1,\text{strict}} \subseteq \mathcal{W}_{1,\text{strict}}) \\
&\geq \frac{C}{6}\Pr(\mathcal{W}_{1,\text{ties}}) && (\text{by Lemma 13}) \\
&\geq \frac{C}{6}\Pr(\mathcal{W}_1) && (\mathcal{W}_1 \subseteq \mathcal{W}_{1,\text{ties}}) \\
&\geq C_6 \Pr(\mathcal{W}_1),
\end{aligned}$$

if we set $0 < C_6 = \min\left(\frac{1}{2}, \frac{C}{6}\right) < 1$. This concludes the proof of Lemma 16. ◀

B Tools

▶ **Lemma 22** (Multiplicative forms of Chernoff bounds [21]). *Let $X_1, \ldots, X_n$ be independent binary random variables. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then:*

1. *For any $\delta \in (0,1)$ and any $\mu \leq \mu_+ \leq n$, it holds that*

$$\Pr(X \geq (1+\delta)\mu_+) \leq \exp\left(-\delta^2\mu_+/3\right).$$

2. *For any $\delta \in (0,1)$ and any $0 \leq \mu_- \leq \mu$, it holds that*

$$\Pr(X \leq (1-\delta)\mu_-) \leq \exp\left(-\delta^2\mu_-/2\right).$$

▶ **Lemma 23** (Hoeffding bounds [33]). *Let $a < b$) be two constants, and $X_1, \ldots, X_n$ be independent binary random variables such that $\Pr(a \leq X_i \leq b) = 1$ for all $i \in [n]$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then:*

1. *For any $t > 0$ and any $\mu \leq \mu_+$, it holds that*

$$\Pr(X \geq \mu_+ + t) \leq \exp\left(-\frac{2t^2}{n(b-a)^2}\right).$$

2. *For any $t > 0$ and any $0 \leq \mu_- \leq \mu$, it holds that*

$$\Pr(X \leq \mu_- - t) \leq \exp\left(-\frac{2t^2}{n(b-a)^2}\right).$$

▶ **Lemma 24** (Bernstein Inequality [21]). *Let $X_1, X_2, \ldots, X_n$ be i.i.d. random variables such that $|X_i - \mathbb{E}[X_i]| \leq C$ and $|\mathbb{E}[X_i]| \leq C$ for some $C > 0$. Define the sum of centered variables $S_n = \sum_{i=1}^{n}(X_i - \mathbb{E}[X_i])$. Then, for all $t > 0$,*

$$\Pr(|S_n| \geq t) \leq 2\exp\left(-\frac{t^2}{2n\mathrm{Var}(X_1) + \frac{2Ct}{3}}\right).$$

Byzantine Consensus in the Random Asynchronous Model

George Danezis
Mysten Labs, London, UK
University College London, UK

Jovan Komatovic
EPFL, Lausanne, Switzerland

Lefteris Kokoris-Kogias
Mysten Labs, Athens, Greece

Alberto Sonnino
Mysten Labs, London, UK
University College London, UK

Igor Zablotchi
Mysten Labs, Zurich, Switzerland

Abstract

We propose a novel relaxation of the classic asynchronous network model, called the *random asynchronous model*, which removes adversarial message scheduling while preserving unbounded message delays and Byzantine faults. Instead of an adversary dictating message order, delivery follows a random schedule. We analyze Byzantine consensus at different resilience thresholds ($n = 3f + 1$, $n = 2f + 1$, and $n = f + 2$) and show that our relaxation allows consensus with probabilistic guarantees which are impossible in the standard asynchronous model or even the partially synchronous model. We complement these protocols with corresponding impossibility results, establishing the limits of consensus in the random asynchronous model.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases network model, asynchronous, random scheduler, Byzantine consensus

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.28

Related Version *Full Version*: https://arxiv.org/abs/2502.09116

1 Introduction

Byzantine Fault-Tolerant (BFT) consensus is a fundamental primitive in distributed computing, serving as the backbone of many applications, including blockchains [6]. Solving BFT consensus in an asynchronous network model [13] is a well-studied problem [9, 16, 17, 24]. Typically, algorithms in this model assume that the adversary controls (1) a subset of faulty processes and (2) message delays, particularly the *message schedule* – the order in which messages are delivered.

However, asynchronous BFT consensus is constrained by restrictive lower bounds [13, 19]. These lower bounds often stem from the adversary's ability to impose arbitrary message schedules. Requiring an algorithm to function under the worst possible scheduling scenario often renders tasks impossible due to a single, highly unlikely counterexample, that is, a pathological schedule that violates the algorithm's properties [13].

Yet, due to performance considerations, practical implementations of consensus circumvent the limitations of asynchrony by assuming that the schedule is not adversarial. For example, we are aware of asynchronous consensus protocols running in production over commodity


© George Danezis, Jovan Komatovic, Lefteris Kokoris-Kogias, Alberto Sonnino, and Igor Zablotchi;
licensed under Creative Commons License CC-BY 4.0



wide-area networks for extensive periods of time without a random coin implementation [11] – so as to avoid the performance overhead of cryptographically-secure randomness – without loss of liveness. This motivates our search for a model to explain this empirical observation.

Another common relaxation that circumvents the limitations of asynchrony is to assume *periods of synchrony*, leading to the widely used *partially synchronous* model [12]. However, protocols based on partial synchrony lose liveness outside of these periods, which may occur due to poor network conditions or denial-of-service (DoS) attacks [15].

In this paper, in order to provide a sound basis for non-adversarial scheduling as assumed by some modern practical systems, and to avoid the limitations of partial synchrony, we propose an alternative relaxation of the asynchronous model: we study asynchronous networks without adversarial scheduling. Specifically, we ask: *What becomes possible in a network where message delays are unbounded, but the message schedule is not adversarial?*

To explore this question, we introduce a new variant of the asynchronous model, which we call the *random asynchronous model*. In this model, message delays remain unbounded, and the adversary still controls a subset of processes. However, the message schedule is no longer adversarial; instead, messages are delivered in a *random* order. Our aim is not to argue that this model is a more realistic description of real networks; rather, we view it as a useful abstraction, inspired by assumptions made in practice [11], that lets us analyze the non-adversarial schedule assumption and explore which tasks it enables under unbounded delays while preserving Byzantine faults. This work offers a new perspective on the role of adversarial scheduling in the standard asynchronous setting: by removing the adversary's scheduling power, we explore what becomes achievable and what remains fundamentally impossible. Our work is a part of the broader line of research that explores relaxations of the fully adversarial scheduler: from the fair-scheduler abstraction of Bracha and Toueg [8], through Aspnes's noisy scheduling for shared memory [4], to recent DAG-based BFT systems such as Tusk [11] and Mahi-Mahi [16].

To isolate the impact of random scheduling, we consider only deterministic algorithms, meaning processes do not have access to local randomness. We analyze the impact of our model on Byzantine consensus at different resilience thresholds: $n = 3f + 1$, $n = 2f + 1$, and $n = f + 2$, where n is the total number of processes, and up to f processes may be faulty.

We find that our model enables consensus protocols that are impossible under standard asynchrony. The key insight is that the random asynchronous model prevents an adversary from blocking honest parties from communicating indefinitely. In traditional asynchrony, an adversary can delay messages indefinitely to prevent termination (e.g., in the FLP impossibility result [13]). In contrast, in the random asynchronous model, the adversary cannot control the schedule, ensuring that honest parties can exchange messages within a bounded number of steps with high probability.

This last property is reminiscent of what synchrony guarantees, i.e., that any message sent by a correct process is received within a time bound Δ. This leads to the natural question: *How powerful is the random asynchronous model with respect to the standard models – asynchrony, partial synchrony, and synchrony?* We answer this question in Section 8. In brief, we show that the random asynchronous model sits between asynchrony and synchrony in terms of task solvability power, yet perhaps surprisingly, is *incomparable* to partial synchrony. To illustrate the latter separation, consider the following two tasks: (1) deterministic consensus when at least one process may fail by crashing, and (2) Byzantine consensus for $n < 3f + 1$. Task (1) is not solvable in the random asynchronous model, as we show in Section 7, but is solvable in partial synchrony [9]; conversely, task (2) is not solvable in partial synchrony [12] yet becomes attainable in the random asynchronous model with probabilistic safety (Section 5). Finally,

we also show that the random asynchronous model behaves like a variant of synchrony where timing guarantees hold with high probability instead of deterministically, thus confirming the intuitive similarity between the two models.

1.1 Modeling Challenges

Designing an asynchronous model with a non-adversarial schedule presented several challenges. Our initial attempt at a round-based model, while mathematically tractable, proved too rigid, enforcing communication patterns that were overly restrictive. We then explored probabilistic scheduling over entire message schedules, but this approach was unintuitive and impractical for analysis. A more promising approach was to randomly select individual messages for delivery. However, this exposed a critical vulnerability: Byzantine processes could manipulate the scheduling distribution by flooding the system with messages, effectively regaining control over the schedule. Attempts to mitigate this by encrypting messages failed, as the adversary could still infer crucial protocol information from the message patterns and delivery timings. More details about our initial attempts to model random asynchrony can be found in Section A.1.

These challenges led us to our final model: instead of selecting individual messages, the scheduler randomly selects sender-receiver pairs. At each step, a sender-receiver pair (s, r) is chosen, and the earliest pending message from s to r is delivered. This approach prevents Byzantine nodes from biasing the schedule since the probability of a message being delivered depends only on the number of available sender-receiver pairs, not on the volume of messages sent by a single process.

A natural concern is whether removing adversarial scheduling trivializes the consensus problem. We address this in two ways. First, Appendix A.2 presents a naive algorithm that fails to solve consensus when $n \leq 2f + 1$, highlighting a fundamental challenge: even though the random asynchronous model guarantees eventual communication, Byzantine nodes can still equivocate, preventing honest processes from distinguishing between correct and faulty messages. Second, we complement our positive results with corresponding impossibility results, establishing close bounds on what is solvable in the random asynchronous model.

1.2 Our Results

Our key contribution is the introduction of the random asynchronous model, a novel relaxation of the classic asynchronous model that removes adversarial scheduling while preserving unbounded message delays and Byzantine faults. This relaxation allows us to achieve new bounds that were previously impossible under standard asynchrony. We then design BFT consensus protocols in this new model, analyzing their feasibility at different resilience thresholds. Finally, we provide both positive results (demonstrating feasibility) and impossibility results (establishing the model's limits), summarized in Table 1.

2 Related Work

Random scheduling. Similar assumptions to the random asynchronous model have been explored by previous work. In message passing, Bracha and Toueg [8] define the fair scheduler, which ensures that in each message round, there is a non-zero constant probability that every correct process receives messages from the same set of correct processes. Under this scheduler, they proposed deterministic asynchronous binary consensus protocols for crash and Byzantine fault models. More recently, Tusk [11] and Mahi-Mahi [16] employ a form of

Table 1 Guarantees achievable (top) and impossible (bottom) in the random asynchronous model. "Strong" and "weak" validity are defined in Section 3.

n versus f	**Validity**	**Agreement**	**Termination**
	Positive results		
$n = 3f + 1$	Deterministic strong	Deterministic	Almost surely
$n = 2f + 1$	With high probability strong	With high probability	Deterministic
$n = f + 2$	Deterministic weak	With high probability	Deterministic
	Negative results		
$n = 3f + 1$	Deterministic strong	Deterministic	Deterministic
$n = 2f + 1$	Almost surely strong	Almost surely	Deterministic
$n = f + 2$	With high probability strong	With high probability	Deterministic

random scheduling. Their random scheduler can be seen as a special case of ours: they only consider the $n = 3f + 1$ case and assume a standard round-based model with the subset of processes that a process "hears from" in a given round chosen *uniformly* at random among all possibilities. They leverage the random scheduler to increase the probability to commit at each round, and thus to reduce latency, whereas our paper focuses on circumventing impossibility results in standard asynchrony, at different ratios of fault-tolerance. They conduct experiments without randomization on a wide-area network, without observing loss of liveness, which can serve as motivation for our work.

In shared memory, Aspnes [4] defines noisy scheduling, in which the adversary may choose the schedule, but that adversarial schedule is perturbed randomly. Under this assumption, deterministic asynchronous consensus becomes achievable. Also in shared memory, previous work introduce a *stochastic scheduler* [2, 3], which schedules shared memory steps randomly. This line of work shows that many lock-free algorithms are essentially wait-free when run against a stochastic scheduler, because the schedules that would break wait-freedom have negligible probability.

Randomized consensus. A large body of research leverages random coins to circumvent the FLP impossibility result [13], which states that deterministic consensus is impossible in crash-prone asynchronous systems. In this approach, protocols relax deterministic termination to probabilistic termination, assuming processes have access to a source of (cryptographically-secure) randomness that cannot be predicted by the adversary. Randomized consensus protocols employ either local coins [7, 21, 28] – which produce randomness independently and locally at each process, without coordination with other processes – or common coins [10, 11, 16, 22, 23, 25] – which, through the use of coordination and strong cryptographic primitives, ensure that all correct processes receive the same random output with some probability.

Our algorithms for the $n = 3f + 1$ setting resemble existing coin-based random consensus protocols, with the randomness moved from the process logic to the schedule. In fact, all of the coin-based consensus protocols we examined can be transformed, with minor changes, into deterministic (coin-less) algorithms in the random asynchronous model. A natural question, then, is whether our model is equivalent to the standard asynchronous model with coin tosses. It is not: in the standard model, achieving safety *w.h.p.* when $n < 3f$ is impossible, even if processes have access to randomness. This is because of a standard network partition argument: the adversary can partition correct processes into two sets that

never exchange messages, allowing Byzantine to force different decisions in each set. By contrast, our model makes long-lived network partitions occur with negligible probability, allowing safety *w.h.p.* even for $n < 3f$.

Probabilistic quorum systems. A line of work on probabilistic quorum systems [20, 27] relaxes quorum intersection to be probabilistic rather than deterministic, and allows for probabilistic correctness guarantees. However, they are vulnerable to an adversarial scheduler [1]. ProBFT [5] addresses this through the use of verifiable random functions for quorum selection. These works are similar to ours in that correctness is probabilistic rather than deterministic, but their approach focuses on the $n = 3f + 1$ setting, and is mainly aimed at scalability and efficiency (e.g., communication complexity), whereas we aim to circumvent impossibilities in standard asynchrony across various fault tolerance ratios.

3 Model & Preliminaries

Processes. We consider a set Π of n processes, up to f of which may be faulty. We consider the Byzantine-fault model, in which faulty processes may depart arbitrarily from the protocol. Throughout the paper, we assume that correct (non-faulty) processes have deterministic logic, i.e., they do not have access to a source of randomness.

Cryptography. We make assumptions that are standard for Byzantine fault-tolerant algorithms: processes communicate through authenticated channels (e.g., using message authentication codes (MACs)) and have access to digital signatures whose properties cannot be broken by the adversary.

Network. The processes communicate by sending messages over a fully-connected reliable network: every pair of processes p and q communicate over a link that satisfies the *integrity* and *no-loss* properties. Integrity requires that a message m from p be received by q at most once and only if m was previously sent by p to q. No-loss requires that a message m sent from p to q be eventually received by q.

Random asynchrony. Processes send messages by submitting them to the network; a *random scheduler* decides in which order submitted messages are delivered. We assume the scheduler delivers messages one by one, i.e., no two delivery events occur at exactly the same time. For convenience, we use a global discrete notion of time to reflect the sequence of delivery events: time proceeds in discrete steps $0, 1, \ldots$; each time step corresponds to a message delivery event in the system. Processes do not have access to this notion of time (they do not have clocks). Sending a message and performing local computation occur instantaneously, between time steps. Note that this notion of time does not bound message delays: an arbitrary amount of real time can pass between time steps.

We next describe the random scheduler. At any time t, let $P(t) \subseteq \Pi^2$ be the set of pairs of processes (p, q) such that p has at least one *pending message* to q. We say that a message m from p to q is pending at time t if p sends m before t and q has not received m by time t. At each time step t, the random scheduler draws a pair (p, q) at random from $P(t)$ and delivers to q the earliest message from p. We assume that there exists a lower bound $\mathcal{C}(n, f) > 0$ such that, for any p and q, the probability that $(p, q) \in P(t)$ is drawn at time t is at least $\mathcal{C}(n, f)$. In general, $\mathcal{C}(n, f)$ may depend on n and f, but not on t. We assume that scheduling draws do not depend on the content of the message being delivered.

More formally, let $\{X_t\}_{t\geq 0} \in P(t)$ be the random variables that represent the scheduling draws at each step t, and let $\mathcal{F}_t$ be the history (all messages sent/delivered) up to step t. Then our lower bound assumption above can be written as

$$\Pr[X_t = (p,q) \mid \mathcal{F}_t] \;\geq\; \mathcal{C}(n,f) \quad \text{for any} \quad t \geq 0 \tag{1}$$

Consensus. Our algorithms solve two variants of binary Byzantine consensus: strong and weak. Strong Byzantine consensus is defined by the following properties:
Strong Validity If all correct processes propose v, then correct processes that decide, decide v.
Agreement If correct processes p and q decide v and w respectively, then $v = w$.
Termination Every correct process decides some value.

Weak Byzantine consensus [18] has the same agreement and termination properties as the strong variant above, but has a different validity property:
Weak Validity If all processes are correct and propose v, then processes that decide, decide v.

Deterministic and probabilistic properties. The probability of a schedule is the probability of the intersection of all its scheduling steps. We say that an algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *deterministically* if $\mathcal{P}$ holds in every execution of $\mathcal{A}$. Note that any algorithm that ensures a property $\mathcal{P}$ deterministically in the standard asynchronous model, must also ensure $\mathcal{P}$ deterministically in the random asynchronous model. Given an algorithm $\mathcal{A}$ and a property $\mathcal{P}$, we say that a schedule S is *bad for* $\mathcal{P}$ if there exists an execution E of $\mathcal{A}$ with schedule S such that $\mathcal{P}$ does not hold in E. An algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *almost surely (a.s.)* if the total probability of all schedules that are bad for $\mathcal{P}$ is 0. An algorithm $\mathcal{A}$ ensures a property $\mathcal{P}$ *with high probability (w.h.p.)* if the total probability of all schedules that are bad for $\mathcal{P}$ is negligible; in this paper, a probability is negligible if it approaches 0 exponentially with some parameter of the algorithm, for any (fixed) n and f (e.g., the number of communication steps).

4 $n = 3f + 1$: Deterministic Safety, Termination Almost Surely

In this section we solve binary Byzantine consensus with deterministic strong validity and agreement, and termination almost surely.

Our algorithm, shown in Algorithm 1, proceeds in rounds: in each round, processes attempt to decide using a ROUND procedure; if unsuccessful, processes update their estimate and try again in the next round. The ROUND procedure is similar to an adopt-commit object [14, 26]: processes *propose* a value and return a pair (g, v), where v is a value and g is a grade, which can be either COMMIT or ADOPT. If $g =$ COMMIT, then it is guaranteed that all correct processes return the same value v (possibly with different grades). If all correct processes propose the same value v, then all correct processes are guaranteed to return v with a COMMIT grade.

The core of the algorithm is the ROUND procedure, shown in Algorithm 2. There are two types of messages: INIT and ECHO. Processes rely on Byzantine Reliable Broadcast (BRB) [9] for communication.[1] Furthermore, all messages are signed. The algorithm has two phases. In phase 1, each correct process p broadcasts (using BRB) its input value v in an INIT message (line 2), and waits to deliver $n - f = 2f + 1$ INIT messages (line 3). Process

[1] Any asynchronous BRB protocol is also correct in the random asynchronous model.

Algorithm 1 Main consensus algorithm for $n = 3f + 1$: pseudocode at process i.

```
1: procedure PROPOSE(v_i):                                  ▷ v_i ∈ {0, 1}
2:     est_i ← v_i
3:     r_i ← 0
4:     while true:
5:         (g, v) ← ROUND(r_i, est_i)
6:         if g = COMMIT: decide(v)                         ▷ Only once
7:         est_i ← v
8:         r_i ← r_i + 1
```

Algorithm 2 Byzantine ROUND implementation for $n = 3f + 1$: pseudocode at process i.

```
1: procedure ROUND(r, v):
2:     BRB-Broadcast ⟨INIT, r, v⟩                           ▷ Phase 1
3:     Wait to BRB-Deliver ⟨INIT, r, _⟩ from n − f processes
4:     H ← delivered ⟨INIT, r, _⟩ messages
5:     proposal ← majority value in H
6:     BRB-Broadcast ⟨ECHO, r, proposal, H⟩ to all processes  ▷ Phase 2
7:     Wait to BRB-Deliver valid ⟨ECHO, r, _, _⟩ messages from n − f processes
8:     if ∃ v* ≠ ⊥ such that I received ≥ 2f + 1 ⟨ECHO, r, v*, _⟩ messages:
9:         return ⟨COMMIT, v*⟩
10:    else:
11:        v* ← majority value among received ⟨ECHO, r, _, _⟩ messages
12:        return ⟨ADOPT, v*⟩
```

p adopts as its *proposal* for phase 2, the majority value among the received INIT messages (line 5). Then, in phase 2, p broadcasts an ECHO message with p's phase 2 proposal, as well as the $n - f$ (signed) INIT messages that justify v to be the majority phase 1 value (line 6); p waits for $n - f$ valid ECHO messages (line 7). If all delivered ECHO messages are for the same value v^*, then p commits v^* (line 9); otherwise p adopts the majority value (line 12).

We prove the correctness of our algorithm in Section B. The main intuition is that at each round, a favorable schedule can help all correct processes agree (i.e., adopt the same estimate), by causing them to select the same majority value in phase 1 (e.g., by ensuring that they deliver INIT messages from the same set of processes). Since the schedule is random, it has a non-zero chance of being favorable at each round. Thus, almost surely, the schedule will eventually be favorable. This is similar in spirit to random coin based Byzantine Randomized Consensus algorithms [9], in which correct processes choose their next estimate by tossing a coin, if they do not manage to reach agreement in a given round. Here we are relying on the random schedule instead of the random coin. However, as we show later, in our model, consensus is solvable for settings where it is impossible for standard asynchrony even when equipped with a common coin.

Crash-fault tolerant consensus. A similar approach can be used to solve crash-fault tolerant consensus with $n = 2f + 1$, with the same guarantees of deterministic safety, as well as termination almost surely. Our algorithm uses the same round-based structure in Algorithm 1 and a modified ROUND procedure, that does not require reliable broadcast, and employs standard point-to-point messages instead. The main intuition is similar to our Byzantine algorithm: the random scheduler eliminates the need for a common coin by ensuring that correct processes eventually deliver messages in a favorable order which leads them to agree and thus terminate. Section C gives our algorithm and proves its correctness.

5 $n = 2f + 1$: Deterministic Termination, Safety with High Probability

In this section, we pose $n = 2f + 1$ and solve strong Byzantine consensus such that safety (validity and agreement) holds with high probability and termination is deterministic.

Algorithm 3 shows our proposed protocol. The main idea is as follows: There are $f + 1$ phases, each consisting of R communication rounds, where R is a parameter. R is large enough so that correct processes hear from each other at least once within R rounds with high probability. In each round, a correct process sends its set of *accepted values* (V_i in Algorithm 3) to all other processes. A correct process i *accepts* a value from process j at phase p ($p = 1, \ldots, f + 1$) if (1) j is the origin of v (i.e., the first signature on v is by j), (2) i has not already accepted a value from j, and (3) v has valid signatures from p distinct processes. We say that i accepts a value v at phase p if p is the earliest phase at which i accepts v (from any process).

After the $f + 1$ phases are over, a correct process decides on the majority value within its set of accepted values (i.e., the value that appears most often). Note that at the end of the communication phases, each correct process must have at least one accepted value, since correct processes sign and send their input value in phase 1 and the network is reliable.

The main intuition behind this protocol is that, if a correct process accepts a value v, then all correct processes will accept v by the end of the execution, and thus all correct processes will have the same set of accepted values. Therefore, all correct processes can use a deterministic rule to decide the same value.

Algorithm 3 Binary Byzantine consensus for $n = 2f + 1$; pseudocode for process i.

```
1:  Local variables:
2:    V_i = ∅, a map from processes to signed values

3:  procedure Propose(v):
4:    V_i[i] ← Sign(v)
5:    for phase in 1 ... f + 1:
6:      for round in 1 ... R:
7:        Send (V_i, phase, round) to all processes
8:        valid ← 0
9:        while valid < n − f:
10:         Receive a (V_j, p, r) message ▷ receive single msg per process, phase and round
11:         if ∃v ∈ V_j : v is signed by phase distinct processes and V_i[Origin(v)] = ∅
12:           V_i[Origin(v)] ← Sign(v)
13:         if (p, r) = (phase, round)
14:           valid ← valid + 1
15:   decide MajorityValue(V_i)
```

We now prove that Algorithm 3 satisfies the consensus properties in Section 3.

▶ **Lemma 1.** *With high probability, every correct process receives at least one message from every other correct process in each phase.*

Proof. We start by fixing two correct processes p and q and upper bounding the probability that q does not receive any message from p after R iterations. The probability of not delivering p's round-1 message to q is at most $(1 - \mathcal{C}(n, f))$ at each time step. There must be

at least $R(n-f)$ time steps for q to complete R iterations, since q waits for at least $n-f$ messages at each iteration, and each message received consumes one time step. Thus, using assumption (1), the probability of q not observing p's message is at most

$$(1-\mathcal{C}(n,f))^{R(n-f)} \approx e^{-R\mathcal{C}(n,f)(n-f)}.$$

To finish the proof, we upper-bound the probability of any correct process not observing the input value of some other correct process. We first compute the expected number of (ordered) pairs of processes (p,q) such that q does not observe the input value of p at the end of the R iterations. There are $n(n-1)$ possible pairs, so this expected value is at most $E = n(n-1)e^{-R\mathcal{C}(n,f)(n-f)}$. Now, by Markov's inequality, we have that

$$\Pr(\text{at least one unreachable pair}) \leq E = n(n-1)e^{-R\mathcal{C}(n,f)(n-f)}.$$

For fixed n and f, this probability approaches 0 exponentially in R. ◀

▶ **Lemma 2.** *With high probability, every correct process accepts the input values of every other correct process.*

Proof. By Lemma 1, every correct process receives at least one message from every other correct process in the first phase, w.h.p. Since messages from correct processes always contain their input values with a valid signature, every correct processes i will accept the input value of another correct process j when i receives the first message from j. ◀

▶ **Theorem 3.** *With $n = 2f+1$, Algorithm 3 satisfies strong validity w.h.p.*

Proof. Assume that all correct processes propose the same value v. Then, by Lemma 2, all correct processes will accept at least $n-f = f+1$ copies of value v w.h.p. Since $f+1$ is a majority out of a maximum of $n = 2f+1$ accepted values, correct processes decide v w.h.p. ◀

▶ **Lemma 4.** *If a value v is accepted by a correct process i at phase $p \leq f$, then all correct processes accept v by the end of phase $p+1$, w.h.p.*

Proof. If i accepts v at phase p, then v must have signatures from at least p processes. Process i is not one of the p processes, otherwise i would have accepted v at an earlier phase. Since i accepts v, i adds its signature to v and will send v, as part of V_i, to all processes in phase $p+1$. By Lemma 1, all correct processes will thus receive v by the end of phase $p+1$ w.h.p., and will accept v (if they haven't already), since v has the required number of signatures. ◀

▶ **Lemma 5.** *If a value v is accepted by a correct process i at phase $f+1$, then all correct processes accept v by the end of phase $f+1$, w.h.p.*

Proof. If i accepts v at phase $f+1$, then v must have signatures from at least $f+1$ processes; i is not among these processes, otherwise i would have accepted v at an earlier phase. Since there are at most f faulty processes, v must have at least one signature from a correct process $j \neq i$.

So j must have accepted v at an earlier phase $p \leq f$ and therefore, by Lemma 4, all correct processes will accept v by the end of phase $f+1$ w.h.p. ◀

▶ **Theorem 6.** *With $n = 2f+1$, Algorithm 3 satisfies agreement w.h.p.*

Proof. By Lemma 4 and Lemma 5, with high probability, correct processes have the same set of accepted values by the end of phase $f + 1$, and thus decide the same value. ◀

▶ **Theorem 7.** *With $n = 2f + 1$, Algorithm 3 satisfies deterministic termination.*

Proof. Follows immediately from the algorithm: correct processes only execute for $R(f + 1)$ rounds. In each round, a correct process waits to receive $n - f$ messages from that round, which is guaranteed to occur in a finite number of steps, since there are at least $n - f$ correct processes and the network is reliable (no-loss property). ◀

6 $n = f + 2$: Deterministic Termination and Validity, Agreement w.h.p.

Interestingly, for $n = f + 2$, we can solve weak Byzantine consensus with deterministic validity and termination, and agreement *w.h.p.*, using the same protocol in Algorithm 3.

Weak validity is clearly preserved: if all processes are correct and have the same input value v, no other value is received by any process, and thus all processes decide v.

▶ **Theorem 8.** *With $n = f + 2$, Algorithm 3 satisfies deterministic weak validity.*

Proof. If all processes are correct and propose the same value v, then all $(V_i, phase, round)$ messages will have v as their value, so no process can decide any other value. ◀

▶ **Theorem 9.** *With $n = f + 2$, Algorithm 3 satisfies agreement w.h.p.*

Proof. Lemmas 1, 2, 4, and 5 still hold: their proofs are also valid if $n = f + 2$. Thus the proof of this theorem is the same as the proof of Theorem 6: By Lemma 4 and Lemma 5, with high probability, correct processes have the same set of accepted values by the end of phase $f + 1$, and thus decide the same value. ◀

▶ **Theorem 10.** *With $n = f + 2$, Algorithm 3 satisfies deterministic termination.*

Proof. Correct processes only execute for $R(f + 1)$ rounds. In each phase and round, a correct process waits to receive $n - f$ valid messages from that phase and round. This wait is guaranteed to terminate since there are at least $n - f$ correct processes, correct processes can always produce a valid message (a message (V_i, p, r) is valid if p and r are equal to the current phase and round, respectively), and the network is reliable (no-loss property). ◀

7 Negative Results

In this section we provide negative results that closely match our positive results from previous sections. Intuitively, we show that in the random asynchronous model it is not possible to obtain Byzantine consensus protocols with more powerful guarantees than the protocols we propose in this paper. This shows that the random asynchronous model, while avoiding some restrictions and impossibilities of the standard asynchronous model, is not overly permissive. Concretely, we prove the following three results.

▶ **Theorem 11.** *No protocol can solve consensus in the random asynchronous model with deterministic strong validity, agreement, and termination, if at least one process may fail by crashing.*

▶ **Theorem 12.** *With $n = 2f + 1$, no protocol for Byzantine consensus in the random asynchronous model can ensure strong validity and agreement with probability 1, as well as deterministic termination.*

▶ **Theorem 13.** *With $n = f + 2, f \geq 2$, no protocol for Byzantine consensus in the random asynchronous model can ensure strong validity and agreement with high probability, while ensuring deterministic termination.*

Proof sketch for Theorem 11. This result is equivalent to the FLP impossibility [13] in the random asynchronous model, and the FLP proof holds in our model as well. Essentially, if at least one process can fail by crashing, there exists an infinite bivalent execution (the same execution as constructed in the FLP proof), which prevents processes from deciding without breaking agreement. ◀

Instead of proving Theorem 12 directly, we can prove the following stronger result, which also implies Theorem 12.

▶ **Theorem 14.** *No protocol for consensus in the random asynchronous model can ensure strong validity and agreement almost surely, as well as deterministic termination, if at least one process may fail by crashing.*

Proof. Assume such a protocol $\mathcal{A}$ exists. Since $\mathcal{A}$ ensures deterministic termination, it must terminate in finite time in every execution. Furthermore, $\mathcal{A}$ ensures strong validity and agreement almost surely, so the probability of schedules in which $\mathcal{A}$ terminates in a bivalent state (in the sense of FLP [13]) must be 0. Yet, following the FLP proof, there exists an infinite bivalent execution E. It follows that $\mathcal{A}$ must decide in a finite prefix π of E, when the state is still bivalent. Since π has finite length, its schedule has non-zero probability. We have shown that $\mathcal{A}$ does not ensure agreement and validity almost surely, a contradiction. ◀

Proof of Theorem 13. Assume towards a contradiction that such a protocol $\mathcal{A}$ exists. Call an execution E of $\mathcal{A}$ *heterogeneous* if in E at least 2 processes propose 0 and at least 2 processes propose 1. Assume *wlog* that p_1 and p_2 propose 0, while p_{n-1} and p_n propose 1.

Let E be a heterogeneous execution in which all processes are correct. Let S be the schedule of E. Let π be shortest prefix of S in which all processes have decided; in other words, we truncate any steps in S after the processes have decided. The prefix π must be finite since $\mathcal{A}$ ensures deterministic termination.

We now describe three executions E_1, E_2, and E_3, with the same schedule π, and show that $\mathcal{A}$ must break either strong validity or agreement in one of the executions. This is sufficient to show that π is bad (in the sense of Section 3) for strong validity or agreement.

Let E_1 be an execution with schedule π, in which all processes behave identically to E; in E_1, p_1 and p_2 are correct, while $p_3, \ldots, p_n$ are Byzantine but behave correctly. By strong validity, p_1 must decide 0 in E_1.

Let E_2 be an execution with schedule π, in which again all processes behave identically to E; this time, p_1 and p_n are correct, while $p_2, \ldots, p_{n-1}$ are Byzantine. Since E_1 and E_2 are indistinguishable to p_1, and p_1 has deterministic logic, p_1 must decide the same value in both executions, namely 0. Thus, in order to satisfy agreement, p_n must also decide 0 in E_2.

Let E_3 be an execution with schedule π, in which again all processes behave as in E; this time, p_{n-1} and p_n are correct, while $p_1, \ldots, p_{n-2}$ are Byzantine. Since E_2 and E_3 are indistinguishable to p_n, and p_n has deterministic logic, p_n must decide the same value in both executions, namely 0. But this breaks strong validity, as both correct processes (p_{n-1} and p_n) have proposed 1 in E_3.

We have shown that π is bad for agreement or strong validity. Since the length of π is fixed (and finite) for fixed n and f, the probability of π is non-negligible according to our definition in Section 3. Therefore, $\mathcal{A}$ does not ensure strong validity and agreement with high probability, a contradiction. ◀

8 The Random Asynchronous Model vs Standard Models

In this section we compare the random asynchronous model with the standard network models – asynchrony, synchrony, and partial synchrony – in terms of task solvability. We show that it is **(i) strictly stronger** than full asynchrony (i.e., the set of tasks solvable in random asynchrony is a strict superset of the set of tasks solvable in asynchrony), **(ii) strictly weaker** than synchrony, and **(iii) incomparable** with partial synchrony, with respect to task solvability. Table 2 summarizes the relationships.

Table 2 Relative power of the random asynchronous model.

	Asynchrony	Partial synchrony	Synchrony
Random asynchrony	strictly stronger	incomparable	strictly weaker

Asynchrony. Every task solvable in asynchrony is also solvable in random asynchrony because *every finite prefix of an asynchronous schedule occurs with non-zero probability* under our scheduler.[2] Conversely, the task "Byzantine binary consensus with $n = 2f + 1$, deterministic termination, and safety holding with high probability" is solvable in the random asynchronous model (Section 5) yet impossible in pure asynchrony by the standard network partition argument [7, 8]. Hence the random asynchronous model is strictly stronger.

Synchrony. The random asynchronous model is strictly more restrictive than synchrony: any task that is solvable in the random asynchronous model is also solvable in synchrony, and there are tasks which are solvable in synchrony but not in the random asynchronous model. As an example of the latter, consider deterministic crash-fault tolerant consensus: this task is impossible in the random asynchronous model (Theorem 11) but it is solvable in synchrony.

We now prove the former direction: any task solvable in the random asynchronous model is also solvable in synchrony. Take any task τ and let $\mathcal{A}_{RA}$ be a protocol that solves τ in the random asynchronous model. We can simulate $\mathcal{A}_{RA}$ in synchrony using an adapter Σ_p for each process p between $\mathcal{A}_{RA}$ and the network, in the following way:

1. At each synchronous time step $t = 0, 1, \ldots$, each correct process p executes all outstanding local steps of $\mathcal{A}_{RA}$, including those triggered by messages delivered since the previous time step. Process p sends the same messages it would in $\mathcal{A}_{RA}$, by handing them to Σ_p.
2. The synchronous network delivers all messages send at time t by time $t + 1$. Σ_p buffers any messages received by p between t and $t + 1$.
3. At time $t + 1$, Σ_p draws a fresh random permutation of the links on which p has received messages, then delivers to $\mathcal{A}_{RA}$ the *earliest* pending message on each link in that order.

[2] Formally, let S be any (possibly infinite) asynchronous schedule and π a finite prefix of S. Since each delivery step is chosen independently with lower-bound probability $C(n, f) > 0$, the probability of π is at least $C(n, f)^{|\pi|}$.

The resulting schedule visible to $\mathcal{A}_{RA}$ is valid in the random asynchronous model, as any causal relation is preserved. Since the Σ_p adapters collectively deliver messages from at most n^2 links at each synchronous time step, each link has a probability of at least $1/n^2 := \mathcal{C}(n, f)$ of being selected at each time step internal to the random asynchronous protocol $\mathcal{A}_{RA}$. Thus, the Σ_p adapters preserve the probabilistic delivery properties of the random asynchronous model. We have described a construction that simulates any random asynchronous algorithm in synchrony, thus showing that any task that is solvable in the random asynchronous model is solvable in synchrony.

Partial synchrony. The random asynchronous model is incomparable with respect to partial synchrony in terms of solvability: there are tasks that are solvable in the random asynchronous model, but not in partial synchrony, and vice-versa.

As an example of a task that is solvable in the random asynchronous model but not in partial synchrony, recall the same $n = 2f + 1$ Byzantine consensus task above. Section 5 shows that this task is solvable in the random asynchronous model. However, this task is not solvable in partial synchrony [12].

To show that some tasks are solvable in partial synchrony but not in the random asynchronous model, consider the task of (deterministic) crash-fault tolerant consensus. This task is solvable in partial synchrony if $n \geq 2f + 1$ [9], but is not solvable in the random asynchronous model if at least one process can crash (Theorem 11).

Discussion. Although the random–asynchronous model is *strictly* weaker than full synchrony in terms of task solvability, it can approximate synchronous behavior with high probability. After R rounds of all-to-all communication, every pair of correct processes has exchanged at least one message with probability $1-\exp(-\Omega(R))$ (Lemma 1). This mirrors the deterministic guarantee offered by a single synchronous round – modulo the known time bound Δ.

Leveraging this property, we can implement, in the random asynchronous model, a crash-fault detector $\sim\mathcal{P}$ that is *perfect with high probability*:

Strong completeness Every process that crashes is eventually permanently suspected by every correct process.

Strong accuracy w.h.p. No correct process is suspected by any correct process w.h.p.

The implementation is a classic heartbeat scheme. Each process broadcasts a heartbeat message once per round; process p suspects q iff it has not received a heartbeat from q in the last R rounds. As we increase R, the probability of false positive suspicion decreases exponentially, through an argument similar to Lemma 1.

Such a failure detector is similar to the perfect $\mathcal{P}$ and eventually perfect $\diamond\mathcal{P}$ failure detectors, which can be implemented in synchrony and partial synchrony respectively [9]. Whether protocols that rely on $\mathcal{P}$ or $\Diamond\mathcal{P}$ can be systematically translated to use $\sim\mathcal{P}$ – preserving their guarantees w.h.p.– remains an open question.

Intuitively, such a systematic translation seems unlikely to exist, as downgrading deterministic communication guarantees to hold w.h.p. could break arbitrary internal algorithm invariants. Consider a synchronous algorithm that runs for T rounds; such an algorithm is able to rely on the fact that it will receive messages from the same set of at least $n - f$ correct processes in each of the T rounds – e.g., at the end of round T each process p may output the $(n - f)^{\text{th}}$ highest process id p heard from in every one of the T rounds. However, if we directly translate this algorithm to random asynchrony, by replacing each synchronous round with R asynchronous rounds (for at total of RT rounds), the guarantee of receiving messages from a common subset of $n - f$ processes holds merely with high

probability, not deterministically. In the rare executions where p hears from a common subset of fewer than $n - f$ processes, the translated algorithm would try to access data that does not exist, violating its invariants. Any translation from synchrony to random asynchrony would therefore need manual, algorithm-specific fallback logic to cope with those low-probability outliers, making a generic translation unlikely.

9 Conclusion

We introduce the random asynchronous model, a novel relaxation of the classic asynchronous model that replaces adversarial message scheduling with a randomized scheduler. By eliminating the adversary's ability to indefinitely delay messages, our model circumvents traditional impossibility results in asynchronous Byzantine consensus while preserving unbounded message delays and tolerating Byzantine faults. Our approach avoids the need for synchronized periods (as in partial synchrony) or cryptographic randomness (as in randomized consensus), offering a foundation for practical alternatives to existing asynchronous systems. We demonstrated that this relaxation enables new feasibility results across different resilience thresholds: deterministic safety and probabilistic termination for $n = 3f + 1$, deterministic termination with safety holding with high probability (w.h.p.) for $n = 2f + 1$, and weak validity with w.h.p. agreement for $n = f + 2$. These results are complemented by impossibility bounds, showing our protocols achieve near-optimal guarantees under the model.

Future work could explore extensions of this model to other distributed computing problems, such as state machine replication, and investigate empirical performance trade-offs in real-world deployments. By bridging the gap between theoretical impossibility and practical assumptions, we believe our model opens avenues for more efficient and resilient distributed protocols.

References

1 Amitanand S. Aiyer, Lorenzo Alvisi, and Rida A. Bazzi. On the availability of non-strict quorum systems. In *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724, pages 48–62. Springer, 2005. doi:10.1007/11561927_6.

2 Dan Alistarh, Keren Censor-Hillel, and Nir Shavit. Are lock-free concurrent algorithms practically wait-free? *J. ACM*, 63(4):31:1–31:20, 2016. doi:10.1145/2903136.

3 Dan Alistarh, Thomas Sauerwald, and Milan Vojnovic. Lock-free algorithms under stochastic schedulers. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 251–260. ACM, 2015. doi:10.1145/2767386.2767430.

4 James Aspnes. Fast deterministic consensus in a noisy environment. *J. Algorithms*, 45(1):16–39, 2002. doi:10.1016/S0196-6774(02)00220-1.

5 Diogo Avelas, Hasan Heydari, Eduardo Alchieri, Tobias Distler, and Alysson Bessani. Probabilistic byzantine fault tolerance. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 170–181. ACM, 2024. doi:10.1145/3662158.3662810.

6 Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *AFT*, 2019.

7 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

8 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985. doi:10.1145/4221.214134.

9 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

10 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, pages 123–132. ACM, 2000. doi:10.1145/343477.343531.

11 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

12 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

13 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983*, pages 1–7. ACM, 1983. doi:10.1145/588058.588060.

14 Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, June 28 - July 2, 1998*, pages 143–152. ACM, 1998. doi:10.1145/277697.277724.

15 Giacomo Giuliari, Alberto Sonnino, Marc Frei, Fabio Streun, Lefteris Kokoris-Kogias, and Adrian Perrig. An Empirical Study of Consensus Protocols' DoS Resilience. In *ACM ASIACCS*, 2024.

16 Philipp Jovanovic, Lefteris Kokoris Kogias, Bryan Kumara, Alberto Sonnino, Pasindu Tennage, and Igor Zablotchi. Mahi-mahi: Low-latency asynchronous bft dag-based consensus. *arXiv preprint arXiv:2410.08670*, 2024. doi:10.48550/arXiv.2410.08670.

17 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All You Need is DAG. In *ACM PODC*, 2021.

18 Leslie Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, 1983. doi:10.1145/2402.322398.

19 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

20 Dahlia Malkhi, Michael K. Reiter, Avishai Wool, and Rebecca N. Wright. Probabilistic quorum systems. *Inf. Comput.*, 170(2):184–206, 2001. doi:10.1006/INCO.2001.3054.

21 Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. RITAS: services for randomized intrusion tolerance. *IEEE Trans. Dependable Secur. Comput.*, 8(1):122–136, 2011. doi:10.1109/TDSC.2008.76.

22 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $o(n^2)$ messages, and $o(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015. doi:10.1145/2785953.

23 Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.48.

24 Zhijie Ren, Kelong Cong, Johan Pouwelse, and Zekeriya Erkin. Implicit Consensus: Blockchain with Unbounded Throughput, 2017. arXiv:1705.11046.

25 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

26 Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 297–306. ACM, 1998. doi:10.1145/277697.277755.

27 Haifeng Yu. Signed quorum systems. *Distributed Comput.*, 18(4):307–323, 2006. doi:10.1007/S00446-005-0133-8.

28 Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. Waterbear: Practical asynchronous BFT matching security guarantees of partially synchronous BFT. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 5341–5357. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-haibin.

APPENDIX

A Challenges

A.1 Modeling Challenge

Our aim is to propose a model for asynchrony without adversarial scheduling that is (1) general, i.e., does not restrict algorithm design (2) easy to work with for proofs, (3) usable by practical algorithms, and (4) intuitive. In the course of defining the current model, we came up with several other possibilities that do not meet the aims above:

1. A round-based model, similar to the fair scheduler model of Bracha and Toueg [8]. In each communication round, a correct process sends a message to every process and waits to hear back from $n - f$ processes. The random scheduler assumption is: in each round, a correct process has a non-trivial (i.e., lower-bounded by a constant) probability of hearing from any subset of $n - f$ processes. This model has the advantage of being easy to work with, but is too restrictive, as it restricts algorithms to the round-based structure.
2. A model which places probability directly on entire schedules, instead of on individual communication steps: each valid schedule has a non-trivial probability of occurring. We found this model to be un-intuitive and difficult to work with.
3. A model in which the next message to be delivered is drawn, according to some distribution, from all currently pending messages (i.e., messages that have been sent but not yet delivered). The distribution must ensure that every message has a non-trivial probability to be scheduled next. This model is general (does not restrict algorithm structure), intuitive, and easy to work with, but has the following crucial flaw. Byzatine processes can skew the scheduling distribution by producing a large number of messages (potentially under the guise of retransmitting them as part of the reliable links assumption). If, at any given time, most pending messages are from Byzantine processes, then these messages are more likely to be delivered first, effectively reverting the model to an adversarial scheduler.
4. Similarly to the previous proposal: at each scheduling step, a sender-receiver pair (s, r) is drawn *uniformly at random*, and the earliest pending message from s to r is the next message delivered in the system. This model is intuitive, and easy to work with, while also fixing the message injection attack by Byzantine processes: the number of pending messages from a (potentially Byzantine) process s to process r does not influence the probability distribution of s's messages to be delivered before other messages. The only drawback is with respect to generality: the uniform distribution on the sender-receiver pairs is a strong assumption.

Our final model is similar to the last proposal above, while solving the generality problem by removing the uniform distribution assumption. Instead, we simply assume that the probability of each sender-receiver pair being drawn is non-negligible.

A.2 Algorithmic Challenge

Take the following naive (and incorrect) binary consensus algorithm for the $n \leq 2f + 1$ cases ($n = 2f + 1$ and/or $n = f + 2$):

- Round 0: Processes initially sign and send their input value to all other processes, and wait for such messages from $n - f$ processes.
- Rounds 1–R (where R is a parameter): Processes sign and send their entire history of sent and received messages to all processes and wait for valid such messages from $n - f$ processes.
- At the end of round R, correct processes decide on, say, the lowest input value they have received.

This algorithm is subject to the following attack:

- Assume all correct processes have 1 as their input value.
- The f Byzantine processes do not send any messages to the $n - f$ correct processes up until and including round $R - 1$. Otherwise, Byzantine processes act as correct processes whose input values are 0, including accepting messages from correct processes.
- Thus, no correct process is aware of 0 as a valid input value before round R.
- Let p be some correct processes that the Byzantine processes have agreed on before the start of the execution. The attack attempts to cause p to decide on a different value than the other correct processes, breaking agreement.
- In round R, Byzantine processes send correctly constructed messages to p, containing their entire communication histories. If the random scheduler delivers even one of these messages to p before the end of the round, then p becomes aware of the input value 0 for the first time in round R (and therefore does not have time to inform the other correct processes).
- After round R, p must decide 0 (being the lowest input value it is aware of), while the other processes must decide 1 (not being aware of 0 as a valid input value), breaking agreement.

This attack succeeds if the random scheduler delivers a message from a Byzantine process to p in round R, among the first $n - f$ messages delivered to p in that round. This has a non-trivial chance of occurring.

This algorithm illustrates the main challenge of designing correct algorithms under the random asynchronous model when $n \leq 2f + 1$: even if the model ensures that all correct processes communicate with each other eventually, Byzantine processes can still equivocate and correct processes do not necessarily know which messages are from correct processes and which are not.

B Proofs for Deterministic Safety Algorithms

In this section we prove the correctness of our algorithm in Section 4. We first prove that the ROUND procedure in Algorithm 2 satisfies the properties below, and then prove that Algorithm 1 solves consensus under Byzantine faults.

Strong Validity If all correct processes propose the same value v and a correct process returns a pair $\langle \text{GRADE}, v' \rangle$, then $\text{GRADE} = \text{COMMIT}$ and $v' = v$.

Consistency If any correct process returns $\langle \text{COMMIT}, v \rangle$, then no correct process returns $(\cdot, v' \neq v)$.

Termination If all correct processes propose, then every correct process eventually returns.

In our proofs we rely on the following properties of Byzantine Reliable Broadcast (BRB) [9]:

BRB-Validity If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BRB-No-duplication Every correct process delivers at most one message.

BRB-Integrity If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BRB-Consistency If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

BRB-Totality If some message is delivered by any correct process, every correct process eventually delivers a message.

▶ **Lemma 15.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies strong validity.*

Proof. If all correct processes propose the same value v, then at least $2f + 1$ processes BRB-broadcast an INIT message for v, and therefore at most f processes BRB-broadcast an INIT message for $1 - v$. Thus v will be the majority value among all INIT messages delivered in phase 1, at all correct processes. Thus all correct processes will BRB-broadcast an ECHO message for v. Furthermore, no Byzantine process can produce a valid ECHO message for $1 - v$, since to do so would require a set of $2f + 1$ INIT message with a majority value of $1 - v$. This is impossible due to the properties of BRB and the fact that at most f processes have BRB-broadcast an INIT message for $1 - v$. So, all valid ECHO messages received by correct processes will be for v, so all correct processes will commit v at line 9. ◀

▶ **Lemma 16.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies consistency.*

Proof. If a correct process p_1 commits v at line 9, then it must have delivered a set S_1 of $2f + 1$ ECHO messages for v at line 7. Take now another process p_2 and consider the set S_2 of $2f + 1$ ECHO messages it delivers at line 7. By quorum intersection, S_1 and S_2 must intersect in at least $f + 1$ messages. By the BRB-Consistency property, these $f + 1$ messages must be identical at p_1 and p_2. Thus p_2 delivers at least $f + 1$ ECHO messages for v, which constitutes a majority of the $2f + 1$ ECHO messages it delivers overall. So if p_2 commits a value at line 9, then it must commit v, and if p_2 adopts a value at line 12, then it must adopt v. ◀

▶ **Lemma 17.** *With Byzantine faults and $n = 3f + 1$, Algorithm 2 satisfies termination.*

Proof. Follows immediately from the algorithm and from the properties of Byzantine Reliable Broadcast. Processes perform two phases; the only blocking step of each phase is waiting for $n - f$ messages (lines 3 and 7). This waiting eventually terminates, by the BRB-Validity property and the fact that there are at least $n - f$ correct processes. ◀

▶ **Theorem 18.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies strong validity.*

Proof. This follows from the strong validity property of the ROUND procedure (Lemma 15): if all correct processes propose v to consensus, then all correct processes propose v to ROUND in the first round, where by Lemma 15, all correct processes commit v, and thus all correct processes decide v at line 6. ◀

▶ **Theorem 19.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies agreement.*

Proof. Let r be the earliest round at which some process decides and let p be a process that decides v at round r. We will show that any other process p' that decides, must decide v.

For p to decide v at round r, ROUND must output (COMMIT, v) in that round. Thus, by the consistency property of ROUND, $\text{ROUND}(r, \cdot)$ must output $(\cdot, v)$ at all correct processes. If $\text{ROUND}(r, \cdot)$ outputs (COMMIT, v) for p', then p' decides v at round r (line 6). Otherwise, all correct processes input v to $\text{ROUND}(r+1, \cdot)$, and by the strong validity property, all processes (including p') will output (COMMIT, v) and decide v at round $r+1$. ◀

▶ **Theorem 20.** *With Byzantine faults and $n = 3f + 1$, Algorithm 1 satisfies termination.*

Proof. We can describe the execution of the protocol as a Markov chain with states $0, \ldots, n - f = 2f + 1$; the system is at state i if i correct processes have estimate (est_i variable) equal to 0 before invoking ROUND. Due to the strong validity property of the ROUND procedure, states 0 and $2f + 1$ are absorbing states. There is a non-zero transition probability from each state (including 0 and $2f + 1$), to state 0 or $2f + 1$, or both (we show this below). Therefore, almost surely, the system will eventually reach one of the two absorbing states and remain there. Once this happens (i.e., once all processes have the same est_i variable), the strong validity property of ROUND ensures that all processes (who have not decided yet) will decide within a round.

It only remains to show that there is a non-zero transition probability from each state to at least one of the absorbing states 0 and $2f + 1$. Consider a state $i \notin \{0, 2f + 1\}$; there is a schedule S with non-zero probability which leads the system from i to 0 or $2f + 1$ in one invocation of ROUND. We consider two cases:

- $i < f + 1$: in this case 0 is the minority value among correct processes. In schedule S, the $n - f$ INIT messages delivered by correct processes at line 3 are all from correct processes. Thus, every correct process sees i 0s and $2f + 1 - i$ 1s; 1 is the majority value, so all correct processes adopt it for phase 2. In phase 2, S again ensures that the $n - f$ ECHO messages delivered by correct processes at line 7 are all from correct processes. Thus, all correct processes see $2f + 1$ ECHO messages for 1 and commit 1, bringing the system to state 0.
- $i \geq f + 1$: in this case 0 is the majority value among correct processes. This case is symmetrical with respect to the previous one: the only difference is that all correct processes adopt 0 (the majority value) at the end of phase 1, and all correct processes deliver $2f + 1$ ECHO messages for 0, thus committing 0 and bringing the system to state $2f + 1$. ◀

C Crash-fault Tolerant Consensus in the Random Asynchronous Model

C.1 Definition

In the crash-fault model, faulty processes may permanently stop participating in the protocol at any time, but otherwise follow the protocol. Crash-fault tolerant consensus is defined by the following properties:

Validity If a process decides v, then v was proposed by some process.

Uniform Agreement If processes p and q decide v and w respectively, then $v = w$.

Termination Every correct process decides some value.

C.2 Algorithm

Our algorithm for crash-fault tolerant consensus uses the same round-based structure, shown in Algorithm 1, as our Byzantine consensus algorithm from Section 4. We use a different implementation of the ROUND procedure, shown in Algorithm 4.

Algorithm 4 Crash-tolerant ROUND implementation: pseudocode at process i.

```
1:  procedure ROUND(r, v):
2:      Send ⟨INIT, r, v⟩ to all processes                          ▷ Phase 1
3:      Wait for n − f ⟨INIT, r, _⟩ messages
4:      if ∃ v* such that I received ≥ f + 1 ⟨INIT, r, v*⟩ messages:
5:          proposal ← v*
6:      else:
7:          proposal ← ⊥
8:      Send ⟨ECHO, r, proposal⟩ to all processes                   ▷ Phase 2
9:      Wait for n − f ⟨ECHO, r, _⟩ messages
10:     if ∃ v* ≠ ⊥ such that I received ≥ f + 1 ⟨ECHO, r, v*⟩ messages:
11:         return ⟨COMMIT, v*⟩
12:     else if ∃ v* ≠ ⊥ such that I received ≥ 1 ⟨ECHO, r, v*⟩ messages:
13:         return ⟨ADOPT, v*⟩
14:     else:
15:         v* ← value in first ⟨INIT, r, _⟩ message received
16:         return ⟨ADOPT, v*⟩
```

The ROUND algorithm consists of two phases. In the first phase, every correct process proposes a value by sending it to all processes (line 2). Then it waits to receive proposals from a quorum of processes (line 3). If a process observes that all responses contain the same phase-one proposal value then it proposes that value for the second phase (line 5). If a process does not obtain a unanimous set of proposals in the first phase, the process simply proposes $\perp$ for the second phase (line 7).

Note that as a result of this procedure, if two processes propose a value different from $\perp$ for the second phase, they propose exactly the same value. Let this value be called v^*.

The purpose of the second phase is to verify if v^* was also observed by enough other processes. After a process receives $n - f$ phase-two messages (line 7), it checks if more than f phase-two proposals are equal to v^*, and if so commits v^* (line 11). A process adopts v^* if it receives v^* in the second phase, but is unable to collect enough v^* values to decide (line 13). Finally, it is possible that a process does not receive v^* in the second phase (either because no such value was found in phase one or simply because it has received only $\perp$ in phase two); in this case the process adopts the fist value it received in phase one (line 15).

As in the Byzantine case, the main intuition is that at each round, a favorable schedule can help processes agree (i.e., adopt the same estimate), by causing them to adopt the same estimate at line 15 (e.g., by ensuring that the first INIT message they receive is from the same process). Since the schedule is random, it has a non-zero chance of being favorable at each round. Thus, almost surely, the schedule will eventually be favorable.

C.3 Proofs

We begin with a few lemmas which establish that the ROUND procedure in Algorithm 4 satisfies these properties:

Integrity If a process returns $(\cdot, v)$, then v was proposed by some process.
Strong Validity If all correct processes propose the same value v and a process returns a pair $\langle \textsc{Grade}, v' \rangle$, then $\textsc{Grade} = \textsc{Commit}$ and $v' = v$.
Consistency If any correct process returns $\langle \textsc{Commit}, v \rangle$, then no process returns $(\cdot, v' \neq v)$.
Termination If all correct processes propose, then every correct process eventually returns.

▶ **Lemma 21.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies integrity.*

Proof. We say that a value v is *valid* if it is the input value of some process. We want to show that processes only return valid values. We observe that (1) INIT messages only contain valid values (line 2), and therefore (2) ECHO messages only contain valid values or $\perp$ (lines 4–8). If a process p returns v at line 11 or 13, then p received at least one ECHO message for v, and thus v is valid by (2) above. If p returns v at line 15, then p received at least one INIT message for v, and thus v is valid by (1) above. ◀

▶ **Lemma 22.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies strong validity.*

Proof. If all processes propose the same value v, then all processes send $\langle \textsc{Init}, v \rangle$ at line 2; all processes receive at least $n/2$ $\langle \textsc{Init}, v \rangle$ messages (since $n - f = f + 1 \geq n/2$); all processes adopt v as their proposal for the second phase and send $\langle \textsc{Echo}, v \rangle$ at line 8; all processes receive $n - f = f + 1$ $\langle \textsc{Echo}, v \rangle$ and return $\langle \textsc{Commit}, v \rangle$. ◀

▶ **Lemma 23.** *If a process p sends $\langle \textsc{Echo}, v \rangle$ at line 8, then no process sends $\langle \textsc{Echo}, v' \rangle$, for any $v' \neq v$.*

Proof. Assume the lemma does not hold. Then p must have received more than $n/2$ $\langle \textsc{Init}, v \rangle$ messages, and some process p' must have received more than $n/2$ $\langle \textsc{Init}, v' \rangle$ for some $v' \neq v$. By quorum intersection, it follows that some process must have sent both an $\langle \textsc{Init}, v \rangle$ and $\langle \textsc{Init}, v' \rangle$ message. This is a contradiction, as processes only send one INIT message at line 2. ◀

▶ **Lemma 24.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies consistency.*

Proof. If process p returns $\langle \textsc{Commit}, v \rangle$, it must do so at line 11, after having received $f + 1$ $\langle \textsc{Echo}, v \rangle$ messages. Therefore, every other process p' that returns, must receive at least one $\langle \textsc{Echo}, v \rangle$ message. If p' also receives $f + 1$ $\langle \textsc{Echo}, v \rangle$ messages, then it returns $\langle \textsc{Commit}, v \rangle$. Otherwise, by Lemma 23, p' cannot receive an ECHO message for any other value $v' \neq v$, so p' returns $\langle \textsc{Adopt}, v \rangle$ at line 13. ◀

▶ **Lemma 25.** *With crash faults and $n = 2f + 1$, Algorithm 4 satisfies termination.*

Proof. This follows immediately from the construction of the algorithm. Processes perform two phases; the only blocking step of each phase is waiting for $n - f$ messages (lines 3 and 9). This waiting eventually terminates, since there are at least $n - f$ correct processes. ◀

Now we can show that the consensus protocol in Algorithm 1 is correct under crash faults.

▶ **Theorem 26.** *With crash faults, Algorithm 1 satisfies validity.*

To prove the theorem, we first prove the following lemma:

▶ **Lemma 27.** *If a process proposes v to the ROUND procedure, then v is the consensus input of some process.*

Proof. We prove the result by induction on the round r. For the base case: If $r = 0$, then all processes input their consensus inputs into ROUND. For the induction case: assume the lemma holds up to $r > 0$, and consider the case of $r+1$. By the induction hypothesis and the integrity property of the ROUND procedure, any output of ROUND$(r, \cdot)$ must be the consensus input of some process. Since the input of ROUND$(r+1, \cdot)$ is the output of ROUND$(r, \cdot)$, the lemma must also hold for the case of $r + 1$. This completes the induction. ◀

Proof of Theorem 26. If a process p decides a value v, then the ROUND procedure must have output (COMMIT, v) at line 6. By Lemma 27 and the integrity property of the ROUND procedure, it follows that v is the consensus input of some process. ◀

▶ **Theorem 28.** *With crash faults, Algorithm 1 satisfies uniform agreement.*

Proof. Let r be the earliest round at which some process decides and let p be a process that decides v at round r. We will show that any other process p' that decides, must decide v.

For p to decide v at round r, ROUND must output (COMMIT, v) in that round. Thus, by the consistency property of ROUND, ROUND$(r, \cdot)$ must output $(\cdot, v)$ at all processes. If ROUND$(r, \cdot)$ outputs (COMMIT, v) for p', then p' decides v at round r (line 6). Otherwise, no other value than v can be input to ROUND$(r + 1, \cdot)$, and by the strong validity property, all processes (including p') will output (COMMIT, v) and decide v at round $r + 1$. ◀

▶ **Theorem 29.** *With crash faults, Algorithm 1 satisfies termination almost surely.*

Proof. We can describe the execution of the protocol as a Markov chain with states $0, \ldots, 2f + 1$; the system is at state i if i processes have estimate (est_i variable) equal to 0 before invoking ROUND. Due to the strong validity property of the ROUND procedure, states 0 and $2f + 1$ are absorbing states. There is a non-zero transition probability from each state, other than 0 and $2f + 1$, to each other state (we show this below). Therefore, almost surely, the system will eventually reach one of the two absorbing states and remain there. Once this happens (all processes have the same est_i variable), the strong validity property of ROUND ensures that all processes (who have not decided yet) will decide within a round.

It only remains to show that there is a non-zero transition probability from each state, other than 0 and $2f + 1$, to each other state. Consider a state $i \notin \{0, 2f + 1\}$ and a state j; there is a schedule S with non-zero probability which leads the system from i to j in one invocation of ROUND. In S, every process receives INIT messages for both 0 and 1 at line 3, and thus all processes send ECHO messages with $\perp$ at line 8. Thus, all processes return at line 15. In S, j processes receive an INIT message with value 0 first, so they adopt 0, and $2f + 1 - j$ processes receive an INIT message with value 1 first, so they adopt 1, bringing the system to state j. ◀

Strong Linearizability Without Compare&Swap: The Case of Bags

Faith Ellen
University of Toronto, Canada

Gal Sela
EPFL, Lausanne, Switzerland

Abstract

Because strongly-linearizable objects provide stronger guarantees than linearizability, they serve as valuable building blocks for the design of concurrent data structures. Yet, many objects that have linearizable implementations from base objects weaker than compare&swap objects do not have strongly-linearizable implementations from the same base objects. We focus on one such object: the *bag*, a multiset from which processes can take unspecified elements.

We present the first lock-free, strongly-linearizable implementation of a bag from interfering objects (specifically, registers and test&set objects). This may be surprising, since there are provably no such implementations of stacks or queues.

Since a bag can contain arbitrarily many elements, an unbounded amount of space must be used to implement it. Hence, it makes sense to also consider a bag with a bound on its capacity. However, like stacks and queues, a bag with capacity b shared by more than $2b$ processes has no lock-free, strongly-linearizable implementation from interfering objects. If we further restrict a bounded bag so that only one process can insert into it, we are able to obtain a lock-free, strongly-linearizable implementation from $O(b+n)$ interfering objects, where n is the number of processes.

Our goal is to understand the circumstances under which strongly-linearizable implementations of bags exist and, more generally, to understand the power of interfering objects.

2012 ACM Subject Classification Computing methodologies → Shared memory algorithms; Theory of computation → Data structures design and analysis; Theory of computation → Concurrent algorithms

Keywords and phrases Strong-Linearizability, Bag, Concurrent Data Structures, Wait-Freedom, Lock-Freedom

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.29

Related Version *Full Version*: `https://arxiv.org/abs/2411.19365` [10]

Funding This work was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC) grant RGPIN-2020-04178.

1 Introduction

Concurrent data structures are often built using linearizable implementations of objects as if they were atomic objects. Although linearizability is the usual correctness condition for concurrent algorithms, there are scenarios where composing linearizable implementations does not preserve desired properties. Namely, an algorithm that uses atomic objects may lose some of its properties when these atomic objects are replaced by linearizable implementations. Specifically, linearizability does not preserve hyperproperties (properties of sets of executions), for example, probability distributions of events for randomized programs and security properties such as noninterference [11, 6]. This can be rectified by using strongly-linearizable implementations [11]. They guarantee that the linearization of a prefix of a concurrent execution is a prefix of the linearization of the whole execution. This means that the linearization of operations in an execution prefix cannot depend on later events in the execution.


© Faith Ellen and Gal Sela;
licensed under Creative Commons License CC-BY 4.0



Attiya, Castaneda, and Enea [3] observed that strongly-linearizable implementations of objects typically use compare&swap objects, which can be used to solve consensus among any number of processes. They provided strongly-linearizable implementations of some objects, such as a wait-free single-writer snapshot object and a wait-free max register from a fetch&add object, and a lock-free, readable, resettable test&set object and a lock-free, readable, fetch&increment object from an infinite array of test&set objects and registers. Both fetch&add and test&set objects are less powerful than a compare&swap object.

In this work, we aim to further explore the power of these well-studied building blocks in the context of strong linearizability. Specifically, we investigate strongly-linearizable implementations of a bag from these primitives. Establishing the existence of such implementations deepens our understanding of the circumstances under which these primitives can be used to achieve strong linearizability.

A *bag* is a multiset to which processes can insert elements and from which they can take unspecified elements. A concurrent bag is an abstraction of the interaction between producers and consumers and is commonly used to distribute tasks among processes for load balancing and improved scalability.

Queues and stacks have strongly-linearizable implementations from compare&swap objects [15, 21, 19]. A bag has a straightforward implementation from a stack or a queue. Therefore, a bag has a strongly-linearizable implementation from compare&swap objects. Although there is a wait-free, linearizable implementation of a stack from registers, test&set objects, and a readable fetch&increment object [1] and there is a lock-free, linearizable implementation of a queue from the same set of objects [16], neither of these implementations is strongly-linearizable. In fact, Attiya, Castañeda, and Hendler [5] proved that no lock-free, strongly-linearizable implementation of a stack or queue from such objects is possible. Attiya, Castañeda, and Enea [3] claimed that the lock-free linearizable implementation of a queue which appears in Figure 5, is a strongly-linearizable implementation of a bag. We give a counterexample to this claim in Appendix A. This was the starting point for our research.

The challenge when implementing a strongly-linearizable bag is to enable an operation that is trying to take an element from the bag to detect when the bag is empty. If elements can reside in multiple locations, it does not suffice for the operation to simply examine these locations one by one, concluding that the bag is empty if no elements are found. The problem is that new elements may have been added to locations after they were examined. Even if there was a configuration during the operation in which the bag was empty, it might not be possible to linearize the operation at this point while maintaining strong linearizability. For example, after this point, in an alternative execution, a value could be added to a location that the operation had not yet examined and the operation could return that value, instead of EMPTY.

We address this challenge in Section 4, modifying the lock-free, linearizable implementation of a queue by adding an additional readable fetch&increment object. This object is used by operations which are inserting elements into the bag to inform operations which are trying to take elements from the bag that the bag is no longer empty. We prove that the resulting algorithm is a strongly-linearizable implementation of a bag.Interestingly, although our algorithm is still a linearizable implementation of a queue, it is not a strongly-linearizable implementation of a queue.

A bag can grow arbitrarily large, so either the number of objects used to implement it or the size of the objects must be unbounded. An alternative is a *b-bounded bag*, which can simultaneously contain up to b elements. Unfortunately, Attiya, Castaneda, and Enea [3] proved that a b-bounded bag shared by more than $2b$ processes has no lock-free, strongly-

linearizable implementation from interfering objects [3]. However, we do provide a lock-free, strongly-linearizable implementation of a restricted version of a b-bounded bag, into which only one process, the *producer*, can insert elements. It uses a bounded number of registers, each of bounded size, and readable, resettable test&set objects. Before presenting this implementation, we give two different implementations of a 1-bounded bag with a single producer, which introduce some of the ideas that appear in our b-bounded bag implementation.

In Section 5, we present a wait-free, linearizable implementation of a 1-bounded bag. Note that it provides a stronger progress guarantee. To keep the space bounded, when the producer inserts an element into the bag, it may reuse objects previously used for other elements. It has to do this carefully, to avoid reusing objects that other processes may be poised to access. This requires delicate coordination between the producer and the consumers.

In Section 6, we present a lock-free, strongly-linearizable implementation of a 1-bounded bag. It combines the mechanism for detecting an empty bag from our lock-free, strongly-linearizable implementation of a bag and the mechanism for reusing objects from our wait-free, linearizable implementation of a 1-bounded bag.

Our lock-free, strongly-linearizable implementation of a b-bounded bag, for any value of b, is presented in Section 7. To enable the producer to detect when the bag is full, we use an approach that is similar to detecting when the bag is empty, but we have to ensure that these two mechanisms do not interfere with one another. In our implementation of a 1-bounded bag in Section 6, the producer determines whether the bag is empty or full by inspecting a single object. This makes this implementation both simpler and more efficient than the special case of our implementation of a b-bounded bag when $b = 1$.

2 Preliminaries

We consider an asynchronous system where processes communicate through operations applied to shared objects. Each type of object supports a fixed set of operations that can be applied to it. For example, a *register* supports `read`(), which returns the value of the register, and `write`(x), which changes the value of the register to x.

Each process can be modelled as a deterministic state machine. A *step* by a process specifies an operation that the process applies to a shared object and the response, if any, that the operation returns. The process can then perform local computation, perhaps based on the response of the operation, to update its state. A process can crash, after which it takes no more steps.

A *configuration* consists of the state of every process and the value of every shared object. In an *initial configuration*, each process is in its initial state and each object has its initial value. An *execution* is an alternating sequence of configurations and steps, starting with an initial configuration. If an execution is finite, it ends with a configuration. The order in which processes take steps is determined by an adversarial scheduler.

The *consensus number* of an object is the largest positive integer n such that there is an algorithm solving consensus among n processes using only instances of this object and registers. A register has consensus number 1. Another example of an object with consensus number 1 is an *ABA-detecting register* [2]. It supports `dWrite`(x) and `dRead`(). When `dWrite`(x) is performed, x is stored in the object. When a process p performs `dRead`(), the value stored in the object is returned, together with an output bit, which is true if and only if process p has previously performed `dRead`() and some `dWrite`() has been performed since its last `dRead`(). We use a restricted version of an ABA-detecting register, where `dWrite`() does not have an input parameter and `dRead`() simply returns the output bit.

A *test&set* object has initial value 0 and supports only one operation, `t&s`(). If the value of the object was 0, this operation changes its value to 1 and returns 0. If the value of the object was 1, it just returns 1. In the first case, we say that the `t&s` was *successful* and, in the second case, we say that it was *unsuccessful*. A *resettable test&set* object is like a test&set object, except that it also supports `reset`(), which changes the value of the object to 0. A *fetch&increment* object supports `f&i`(), which increases the value of the object by 1 and returns the previous value of the object. A *compare&swap* object supports the operation `c&s`(u, v), which checks whether the object has value u and, if so, changes its value to v. It always returns the value of the object immediately before the operation was performed. For any type of object that does not support `read`, its readable version supports `read` in addition to its other operations.

Two operations, op and op', *commute* if, whenever op is applied to the object followed by op', the resulting value of the object is the same as when op' is applied followed by op. Note that the responses to op and op' may be different when these operations are applied in the opposite order. The operation op' *overwrites* the operation op if, whenever op is applied to the object followed by op', the resulting value of the object is the same as when only op' is applied. Note that the response to op' may be different depending on whether op was applied. An object is *interfering* if every two of its operations either commute or one of them overwrites the other. All interfering objects have consensus number at most 2 [13]. Registers are interfering. The test&set, resettable test&set, and fetch&increment objects and their readable versions are all interfering and have consensus number 2.

Both a *stack* and a *queue* are examples of non-interfering objects with consensus number 2 [13]. The value of these objects is an unbounded sequence of elements, which is initially empty. A stack supports `push`(x) and `pop`(), whereas a queue supports `enqueue`(x) and `dequeue`().

A *bag* is a non-interfering object whose value is an initially empty multiset of elements. The number of elements that can be in the multiset is unbounded. The elements are taken from a set of values V that does not include $\perp$. A bag supports two operations, `Insert`(x) where $x \in V$ and `Take`. When `Insert`(x) is performed, the element x is added to the multiset. It does not return anything. If the multiset is not empty, a `Take` operation removes an arbitrary element from the multiset and returns it. In this case, we say that the operation is *successful*. If the multiset is empty, then `Take` returns EMPTY and we say that it is *unsuccessful*. Note that this specification is nondeterministic.

A *b-bounded bag* object is an object whose value is a multiset of at most b elements. If the multiset contains fewer than b elements when `Insert`(x) is performed, the element x is added to the multiset and OK is returned. If the multiset already contains b elements, the value of the bag does not change and FULL is returned. For simplicity, we assume no value repetitions in the bounded bags in our proofs.

An *implementation* of an object O shared by a set of processes P from a collection of objects C provides a representation of O using objects in C and algorithms for each process in P to perform each operation supported by O. In an execution, an operation on an implemented object *begins* when a process performs the first step of its algorithm for performing this operation and is *completed* at the step in which the process returns from the algorithm, with the response of the operation, if any.

A bag has a straightforward implementation from a stack or a queue. The multiset is represented by the sequence, `Insert`(x) is performed by applying `push`(x) or `enqueue`(x) and `Take`() is performed by applying `pop`() or `dequeue`(). Thus a bag has consensus number at most 2.

There is a simple algorithm for solving consensus among 2 processes using two single-writer registers and a bag initialized with one element: To propose the value x, a process writes x to its single-writer register and then performs a `Take`() operation on the bag. If the `Take`() returns an element, the process returns x. Otherwise, it reads and returns the value that the other process wrote to its single-writer register. It follows from Borowsky, Gafni, and Afek's *Initial State Lemma* [7, Lemma 3.2], that consensus among 2 processes can be solved from registers and two initially empty bags. Thus, a bag has consensus number exactly 2.

For any execution, α, consider an ordering of every completed operation and a (possibly empty) subset of the operations that have begun, but have not completed, such that, for every completed operation, *op*, each of these operations that begins after *op* has completed is ordered after *op*. A way to obtain such an ordering is to first assign a linearization point in α to each of these operations within its *operation interval*, i.e., no operation is linearized before it begins or after it has completed. Then specify an ordering of the operations that are linearized at the same point. Suppose there is a sequential execution in which the operations in the ordering are performed in order such that every completed operation in α has the same response as it does in this sequential execution. Then we say that the ordering is a *linearization* of α.

An implementation of an object is *linearizable* [14] if each of its executions has a linearization. An implementation is *strongly-linearizable* [11] if there is a function f that maps each execution α to a linearization of α such that, for every prefix α' of α, $f(\alpha')$ is a prefix of $f(\alpha)$.

An implementation is *wait-free* if, in every execution, every operation by a process that does not crash completes within a finite number of steps by that process. An implementation is *lock-free* if every infinite execution contains an infinite number of completed operations.

3 Related Work

Even though there are wait-free, linearizable implementations of registers, max-registers, and single-writer snapshot objects from single-writer registers, Helmi, Higham, and Woelfel [12] proved that there are no lock-free, strongly-linearizable implementations. They also gave a wait-free, strongly-linearizable implementation of a bounded max-register from registers. Denysyuk and Woelfel [9] proved there are no wait-free, strongly-linearizable implementations of max-registers and single-writer snapshot objects from registers, although they gave lock-free strongly-linearizable implementations. Later, Ovens and Woelfel [20] gave lock-free, strongly-linearizable implementations of an ABA-detecting register and a single-writer snapshot object from a bounded number of bounded size registers.

Li [16] gave a lock-free, linearizable implementation of a queue from a fetch&increment object, an infinite array of test&set objects, and an infinite array of registers. He also gave a wait-free, linearizable implementation of a queue in which at most 2 processes may dequeue. It uses a fetch&increment object and infinite arrays of registers and test&set objects. David [8] gave a wait-free, linearizable implementation of a queue in which at most 1 process may enqueue. It uses registers, an infinite array of fetch&increment objects, and a two-dimensional infinite array of swap objects. It is unknown whether there is a wait-free, linearizable implementation of a queue from interfering objects in which any number of processes can enqueue and dequeue.

Afek, Gafni, and Morrison [1] gave a wait-free, linearizable implementation of a stack from interfering objects (specifically, a fetch&add object, registers, and test&set objects). Attiya and Enea [6] showed that this implementation is not strongly-linearizable. Attiya, Castañeda,

and Enea [3] later proved that there is no lock-free, strongly-linearizable implementation of a stack or a queue shared by more than 2 processes from interfering objects.

Many universal constructions (for example, from compare&swap objects and registers or from consensus objects and registers) provide strongly-linearizable, wait-free implementations of any shared object [11]. Treiber [21, 19] gave a simple lock-free, strongly-linearizable implementation of a stack from compare&swap objects and registers. Michael and Scott [18] gave a lock-free, linearizable implementation of a queue from compare&swap objects and registers. Assuming nodes are never reused, it is possible to make a small modification to their implementation to make it strongly-linearizable [15].

4 A Lock-Free, Strongly-Linearizable Implementation of a Bag

The first algorithm we present is a strongly-linearizable implementation of an unbounded bag. The challenge in designing such an algorithm from interfering objects lies in identifying when the bag is empty. To address this, we use a readable fetch&increment object, `Done`, which an `Insert` operation increments as its last step, to inform `Take` operations that the bag is not empty. A detailed description of the algorithm follows.

The implementation uses an infinite array of registers, $Items$, in which elements that have been inserted into the bag are stored, together with an infinite array of test&set objects, TS. The process that performs a successful `t&s`() on $TS[i]$ returns the element stored in $Items[i]$ and removes it from the bag.

Each `Insert`(x) operation begins by performing `f&i`() on a readable fetch&increment object, `Allocated`. This allocates the operation a new location within `Items` in which it writes x. Thus, the value of `Allocated` is the index of the last allocated location within `Items`. Finally, the operation performs `f&i`() on a second readable fetch&increment object, `Done`, to inform `Take`() operations about the insertion.

A `Take`() operation begins by reading `Done` and `Allocated`. Then it reads the allocated locations in `Items`. For each location, `i`, if `Items[i]` contains $x \neq \perp$, the operation performs `t&s`() on the corresponding test&set object, `TS[i]`. If this is successful, x is returned. After reading all the allocated locations in `Items` without performing a successful `t&s`(), the operation rereads `Done`. If its value has not changed since the operation last read `Done`, EMPTY is returned. Otherwise, the operation repeats the entire sequence of steps. Note that it begins again starting from the first location, in case an `Insert` operation that was allocated an early location has recently written to that location. Pseudocode for our implementation of a bag appears in Figure 1.

Strong-linearizability. Consider any execution consisting of operations on this data structure. We linearize the operations as follows:

- A `Take`() operation that performs a successful `t&s`() on Line 17 is linearized at this step. It returns the element it last read on Line 15.
- A `Take`() operation that reads `Done` on Line 18 and obtains the same value it obtained in its previous read of `Done` (on Line 12) is linearized at this last read. It returns EMPTY.
- Consider an `Insert`(x) operation, *ins*, that was allocated location `m` and wrote x to `Items[m]` on Line 8. If some `Take`() operation performs a successful `t&s`() on `TS[m]` after *ins* performed Line 8, but before *ins* performs `Done.f&i`() on Line 9, then *ins* is linearized immediately before this `Take`(). In this case, we say that these two operations are *coupled*. Otherwise, *ins* is linearized when it performs `Done.f&i`().

```
1  Shared variables:
2     Items[1..]: an infinite array of registers, each initialized to ⊥
3     TS[1..]: an infinite array of test&set objects, each initialized to 0
4     Allocated: a readable fetch&increment object, initialized to 0
5     Done: a readable fetch&increment object, initialized to 0
6  Insert(x):
7     m ← Allocated.f&i() + 1
8     Items[m].write(x)
9     Done.f&i()
10 Take():
11    repeat
12       d ← Done.read()
13       m ← Allocated.read()
14       for i ← 1 to m do
15          x ← Items[i].read()
16          if x ≠ ⊥ then
17             if TS[i].t&s() = 0 then return x
18       if d = Done.read() then return EMPTY
```

Figure 1 A lock-free, strongly-linearizable bag.

The linearization point of each operation occurs within its operation interval. At any configuration, C, the bag contains the difference between the multiset of elements inserted by `Insert` operations linearized before C and the multiset of elements returned by successful `Take` operations linearized before C.

Each `Take`() operation is linearized immediately before it returns. Thus, if it is linearized at some point in an execution, it is linearized at the same point in every extension of that execution. Each `Insert`(x) operation that is coupled with a `Take`() operation is linearized immediately before that `Take`() operation. This means that the inserted element is taken from the bag immediately after it is inserted. Thus, if a coupled `Insert`(x) operation is linearized at some point in an execution, it is linearized at the same point in every extension of that execution. Each uncoupled `Insert`(x) operation is linearized when it increments `Done`. This ensures that, if a `Take`() operation sees that the value of `Done` has not changed between Line 12 and Line 18, then no uncoupled `Insert` has been linearized during this part of the execution. An uncoupled `Insert`(x) operation is linearized immediately before it returns, so it is linearized at the same point in every extension of the execution. Hence, if the implementation is linearizable, it is strongly-linearizable.

To show that the ordering is a linearization, it remains to prove that the values returned by the `Take`() operations are consistent with the sequential specifications of a bag. Let tk be a `Take`() operation that performs a successful `t&s`() of `TS`[i] on Line 17 and let $x \neq \perp$ be the element it read from `Items`[i] when it last performed Line 15. Since tk read x from `Items`[i], there was an `Insert`(x) operation that wrote x into `Items`[i] on Line 8. Note that there was exactly one `Insert` operation that was allocated i on Line 7, by the semantics of `f&i`(). This `Insert`(x) operation was linearized before tk, either because it executed Line 9 before tk performed its successful `t&s`() on `TS`[i] or it was coupled with tk and, hence, linearized immediately before tk. The element x remains in the bag until the linearization point of tk because the element in `Items`[i] is only returned by the `Take`() operation that performs the successful `t&s`() on `TS`[i].

Now let tk be a `Take` operation that reads `Done` on Line 18 and obtains the same value it obtained in its previous read of `Done` on Line 12. Let C be the configuration immediately after tk reads `Done` on Line 12 in its last iteration of the repeat loop and let C' be the

configuration immediately before tk reads `Done` for the last time on Line 18. The value of `Done` does not change between these two configurations. Let m be the value tk read from `Allocated` on Line 13, which is the largest location in `Items` that had been previously allocated to an `Insert` operation.

Any element, x, that was in the bag in configuration C was inserted into the bag by an uncoupled `Insert`(x) operation. This operation was linearized when it performed Line 9. Suppose that it wrote x into `Items`[i]. This happened prior to C. Note that $i \leq m$, since m was obtained when tk read `Allocated` on Line 13 after C and the value of `Allocated` only increases. When tk performed Line 15 between configurations C and C', it read x from `Items`[i]. Since tk later reaches Line 18, its `t&s`() on `TS`[i] was unsuccessful. Some other `Take` operation previously performed a successful `t&s`() on `TS`[i] and, when it did, element x was taken from the bag. Therefore, all elements that were in the bag in configuration C were taken by `Take` operations other than tk before C'.

Now consider any element that was inserted into the bag between configurations C and C' by some `Insert` operation. Since the value of `Done` did not change between C and C', this `Insert` operation did not perform `Done.f&i()` on Line 9. Hence, it was coupled with a `Take` operation and it was immediately taken from the bag after it was inserted into the bag. Therefore, all elements that were inserted into the bag between configurations C and C' were taken from the bag prior to configuration C'. Hence, the bag is empty in configuration C', immediately before the step at which tk is linearized.

Lock-freedom. The `Insert` operation performs 3 steps, so it is wait-free. When a `Take` operation finishes an iteration of the loop and is about to begin another iteration, the value of `Done` it last read on Line 18 was different than what it last read on Line 12. This means that some `Insert` operation performed `Done.f&i()` on Line 9 between these two points of the execution. This step completes the `Insert` operation.

It is worth noting that the algorithm in Figure 1 also implements a linearizable queue. The proof of this fact is very similar to the proof used by Li [16]. However, it is not a strongly-linearizable queue, as we prove in Appendix B.

5 A Wait-Free, Linearizable Implementation of a 1-Bounded Bag with a Single Producer

Our first bounded bag algorithm from interfering objects is a wait-free, linearizable implementation of a bag that can contain at most one element. It is shared by n processes, $P_1, \ldots, P_n$, called *consumers*, that can perform `Take`() and a single process, called the *producer*, that can perform `Insert`(x). To keep the space bounded, the producer reuses locations in `Items` to store new elements. The producer announces the most recent location it has allocated by writing it to the shared register `Allocated`. There is a resettable test&set object, instead of a test&set object, associated with each location in `Items`. An array, `Hazards`, of hazard pointers [17] is used to prevent multiple consumers from returning the same element, and to ensure that each element is consumed before being overwritten by a new element. Each consumer announces a location it is about to access and the producer avoids reusing the announced locations. Specifically, the producer avoids resetting the corresponding test&set objects and writing new elements to the corresponding locations in `Items`.

The producer maintains two persistent local variables, `m`, which is the last location in `Items` it allocated, and `used`, which is a set of locations it has used and will need to reset before they are reallocated. To eliminate the need for a special case to handle the first

```
19 Shared variables:
20    Items[1..n+1]: an array of single-writer registers, all initialized to ⊥,
          which can only be written to by the producer
21    TS[1..n+1]: an array of readable, resettable test&set objects, all
          initialized to 0, except for TS[1], which is initialized to 1
22    Allocated: a single-writer register, initialized to 1, which can only be
          written to by the producer
23    Hazards[1..n]: an array of single-writer registers, all initialized to ⊥,
          where Hazards[i] can only be written to by P_i
24 Persistent local variables of the producer:
25    used: a register, initialized to ∅, which contains a subset of
          {1,...,n+1}
26    m: a register, initialized to 1, which is a local copy of Allocated
27 Insert(x) by the producer:
28    if TS[m].read() = 0 then return FULL
29    Items[m].write(⊥)
30    used ← used ∪ {m}
31    hazardous ← COLLECT(Hazards)
32    m ← some index in {1,...,n+1} - hazardous
33    Allocated.write(m)
34    for all i ∈ used - hazardous do
35       TS[i].reset()
36    used ← used ∩ hazardous
37    Items[m].write(x)
38    return OK
39 Take() by consumer P_i for i ∈ {1,...,n}:
40    a ← Allocated.read()
41    Hazards[i].write(a)
42    x ← Items[a].read()
43    if x ≠ ⊥ then
44       if TS[a].t&s() = 0 then
45          Hazards[i].write(⊥)
46          return x
47    Hazards[i].write(⊥)
48    return EMPTY
```

Figure 2 A wait-free, linearizable 1-bounded bag with one producer and n consumers.

`Insert` call (which is the only one not preceded by a previous insertion), the initial state of the data structure is as if location 1 had been allocated to the producer, it had inserted an element into the bag in this location, and then this element had been taken by some consumer. This is simulated by setting the initial values of `m`, `Allocated`, and `TS[1]` to 1.

The producer begins an `Insert`(x) operation by checking whether the test&set object, `TS[m]`, in the last allocated location, `m`, is 0. If so, it returns FULL. Otherwise, it overwrites the element in `Items[m]` with $\perp$ and adds the index `m` to `used`. Afterwards, it collects the set of *hazardous* locations stored in `Hazards`. It then allocates an arbitrary location from $\{1, \ldots, n+1\}$ that is not hazardous and announces this location by writing it to `Allocated`. Next, it resets the test&set for each location in `used` that is not hazardous. Then it removes these locations from `used`. Finally, it writes x to the newly allocated location in `Items` and returns OK.

To perform a `Take`() operation, a consumer reads `Allocated`. It announces the location it read in `Hazards` and then reads the value in this location. If it is an element $x \neq \perp$, the consumer performs `t&s`() on the resettable test&set object for this location. It then clears its announcement. If the `t&s`() was successful, it returns x. If either $x = \perp$ or the `t&s`() was unsuccessful, it returns EMPTY.

If Items[a] contains an element $x \neq \perp$, but this element has already been taken from the bag, it is essential that no consumer can perform a successful t&s() on the associated test&set object, TS[a], and take the element again. To ensure this, a consumer, P_i writes a to Hazards[i] before reading Items[a]. This prevents the producer from resetting TS[a] while P_i is poised to access TS[a].

After it has set Items[m] to $\perp$, but before resetting any test&set objects, the producer will collect the locations that appear in Hazards and refrain from resetting the test&set objects associated with the hazardous locations it collected. Hence, it will not reset a test&set object which any consumer is poised to access.

Assuming that the universe of elements that can be inserted into the bag is bounded, the registers used in the implementations store bounded values and thus, the amount of space used by the implementation is bounded. Pseudocode for our implementation appears in Figure 2.

Consider any execution consisting of operations on this data structure. We linearize the operations as follows:

- A Take() operation that performs a successful t&s() on Line 44 is linearized at this step. It returns the element it last read on Line 42.
- Consider a Take() operation, tk, that returns EMPTY. If Items[a] $= \perp$ or TS[a] $= 1$ when tk read a from Allocated on Line 40, then tk is linearized at this step. Otherwise, we can show that some other Take() operation performed a successful t&s() on TS[a] after tk performed Line 40, but before tk returned. In this case, tk is linearized immediately after the first such Take() operation.
- An Insert(x) operation that reads 0 from TS[m] (on Line 28) is linearized at this read. It returns FULL.
- An Insert(x) operation that writes x on Line 37 is linearized at this write. It returns OK.

Because the code contains no unbounded loops, the implementation is wait-free. Keeping track of hazardous locations enables objects to be safely reused so that bounded space is used. However, this makes the algorithm and its proof of correctness more intricate. In the full version of the paper [10], we prove that Figure 2 is a linearizable implementation of a 1-bounded bag. Appendix C shows that it is not strongly-linearizable.

6 A Lock-Free, Strongly-Linearizable Implementation of a 1-Bounded Bag with a Single Producer

In this section, we present a lock-free, strongly-linearizable implementation of a bag that can contain at most one element. It is shared by n processes, $P_1, \ldots, P_n$, called *consumers*, that can perform Take() and a single process, called the *producer*, that can perform Insert(x). It combines ideas from our lock-free, strongly-linearizable implementation of an unbounded bag in Section 4 and our wait-free, linearizable implementation of a 1-bounded bag in Section 5. To keep the space bounded, we use an ABA-detecting register for Done, instead of a readable fetch&increment object. An ABA-detecting register has a lock-free, strongly-linearizable implementation from registers using bounded space [20].

An Insert(x) operation by the producer is the same as in the wait-free linearizable implementation presented in the previous section, except the producer writes to the ABA-detecting register Done before it returns OK.

To perform a Take() operation, a consumer, P_i, reads Done and Allocated. In Hazards[i], it announces the location, a, that it read from Allocated. Then P_i reads Items[a] and, if it contains an element $x \neq \perp$, then P_i performs t&s() on TS[a]. Next P_i

```
49 Shared variables:
50    Items[1..n+1]: an array of single-writer registers, all initialized to ⊥,
           which can only be written to by the producer
51    TS[1..n+1]: an array of readable, resettable test&set objects, all
          initialized to 0, except for TS[1], which is initialized to 1
52    Allocated: a single-writer register, initialized to 1, which can only be
          written to by the producer
53    Hazards[1..n]: an array of single-writer registers, all initialized to ⊥,
           where Hazards[i] can only be written to by P_i
54    Done: an ABA-detecting register
55 Persistent local variables of the producer:
56    used: a register, initialized to ∅, which contains a subset of
          {1,...,n+1}
57    m: a register, initialized to 1, which is a local copy of Allocated
58 Insert(x) by the producer:
59    if TS[m].read() = 0 then return FULL
60    Items[m].write(⊥)
61    used ← used ∪ {m}
62    hazardous ← COLLECT(Hazards)
63    m ← some index in {1,...,n+1} - hazardous
64    Allocated.write(m)
65    for all i ∈ used - hazardous do
66       TS[i].reset()
67    used ← used ∩ hazardous
68    Items[m].write(x)
69    Done.dWrite()
70    return OK
71 Take() by P_i, for i ∈ {1,...,n}:
72    Done.dRead()
73    repeat
74       a ← Allocated.read()
75       Hazards[i].write(a)
76       x ← Items[a].read()
77       if x ≠ ⊥ then
78          if TS[a].t&s() = 0 then
79             Hazards[i].write(⊥)
80             return x
81       Hazards[i].write(⊥)
82       if Done.dRead() = false then return EMPTY
```

Figure 3 A lock-free, strongly-linearizable 1-bounded bag with one producer and n consumers.

clears its announcement. If the t&s() was successful, P_i returns x. If either $x = \perp$ or the t&s() was unsuccessful, P_i rereads Done and, if its value has not changed since P_i's previous read, P_i returns EMPTY. Otherwise, P_i repeats the entire sequence of steps. Pseudocode for our implementation appears in Figure 3.

Linearizability. Consider any execution consisting of operations on this data structure. We linearize the operations as follows:

- A Take() operation that performs a successful t&s() on Line 78 is linearized when it performs this t&s(). It returns the value it read in its last execution of Line 76.
- A Take() operation that gets false from Done.dRead() on Line 82 is linearized at this step. It returns EMPTY.
- An Insert(x) operation that reads 0 from TS[m] (on Line 59) is linearized at this read. It returns FULL.

- Consider an `Insert`(x) operation, *ins*, that allocated location `a` to `m` on Line 63 and wrote x to `Items[a]` on Line 68. If some `Take`() operation performs a successful `t&s`() on `TS[a]` after *ins* performed Line 68, but before *ins* performs `Done.dRead()` on Line 69, then *ins* is linearized immediately before the `Take`() operation that performed the first such `t&s`(). In this case, *ins* returns OK and we say that these `Insert`(x) and `Take`() operations are *coupled*.
- An `Insert`(x) operation that performs `Done.dWrite()` on Line 69, but has not already been linearized as a coupled operation, is linearized at this step. In this case, it returns OK and we say that it is *uncoupled*.

The proof that Figure 3 is a lock-free, strongly-linearizable implementation of a b-bounded bag appears in the full version of the paper [10].

7 A Lock-Free Strongly-Linearizable Implementation of a b-Bounded Bag with a Single Producer

Our final algorithm is a lock-free, strongly-linearizable implementation of a bag that can contain at most b elements. It is shared by n processes, $P_1, \ldots, P_n$, called *consumers*, that can perform `Take`() and a single process, called the *producer*, that can perform `Insert`(x). It extends our lock-free, strongly-linearizable implementation of a 1-bounded bag in Section 6. In this algorithm, `Allocated` is a shared register that stores a subset of at most b locations, and `alloc` is a local variable the producer uses to update the contents of `Allocated`. The challenge in designing a strongly-linearizable bag algorithm from interfering objects was for an operation that is trying to take an element from the bag to detect when the bag is empty (and then return EMPTY). This is addressed using an ABA-detecting register, `InsertDone`, to which each `Insert`(x) operation writes after writing x to `Items`. When designing a strongly-linearizable *bounded* bag algorithm from interfering objects, an operation that is trying to insert an element into the bag faces a symmetrical challenge, as it needs to detect when the bag is full (and then return FULL). To address this, we use another ABA-detecting register, `TakeDone`, to which each `Take`() operation writes after performing a successful `t&s`(). Unsuccessful `Take` operations also write to `TakeDone` before returning, to help them be linearized before the unsuccessful `Take`.

The producer begins an `Insert`(x) operation by reading `TakeDone`. Then it looks at each of the locations in `TS` that have been allocated. For each location m that contains 1, it clears the element in that location (setting `Items[m]` to $\perp$), removes `m` from `alloc`, and adds `m` to `used`. If `alloc` contains less than b locations, it adds x to the bag as follows. It first collects the non-$\perp$ values from the `Hazards` array into the set `hazardous`. Afterwards, it allocates an arbitrary location, `m`, from $\{1, \ldots, n+b\}$, excluding those in `alloc` and in `hazardous`, adds this location to `alloc`, and announces it by copying `alloc` into `Allocated`. Next, for each location in `used` that is not in `hazardous`, it resets the associated test&set object and removes the location from `used`. Then it writes x to the allocated object, `Items[m]`. Finally, it writes to the ABA-detecting register `InsertDone` and returns OK. If `alloc` contained b locations, it rereads `TakeDone` and, if its value has not changed since its previous read, it returns FULL; otherwise, it repeats the entire sequence of steps.

To perform a `Take`() operation, a process reads `InsertDone` and `Allocated`. For each location a in the set of locations obtained from `Allocated`, it announces a in `Hazards` and then reads the element, `x` from `Items[a]`. If $x \neq \perp$, it performs `t&s`() on the associated test&set object, `TS[a]`, and, if successful, it clears its announcement, writes to `TakeDone`, and returns x. If the process completes the for loop, it clears its announcement, and

rereads `InsertDone`. If the value of `InsertDone` has not changed since its previous read, it writes to `TakeDone` and returns EMPTY. Otherwise, it repeats the entire sequence of steps. Pseudocode for our implementation appears in Figure 4.

```
83   Shared variables:
84   Items[1..n+b]: an array of single-writer registers, each initialized
         to ⊥, which can only be written to by the producer
85   TS[1..n+b]: an array of readable, resettable test&set objects, each
         initialized to 0
86   Allocated: a single-writer register, initialized to ∅, which contains
         a subset of {1,...,n+b} of size between 0 and b and can only be
         written to by the producer
87   Hazards[1..n]: an array of single-writer registers, each initialized
         to ⊥, where Hazards[i] can only be written to by P_i
88   InsertDone, TakeDone: ABA-detecting registers
89   Persistent local variables of the producer:
90   used: a register, initialized to ∅, which contains a subset of
         {1,...,n+b}
91   alloc: a register, initialized to ∅, which is used to update Allocated
92   Insert(x) by the producer:
93   TakeDone.dRead()
94   repeat
95   for all m ∈ alloc do
96   if TS[m].read() = 1 then
97   Items[m].write(⊥)
98   alloc ← alloc - {m}
99   used ← used ∪ {m}
100  if |alloc| < b then
101  hazardous ← COLLECT(Hazards)
102  m ← some index in {1,...,n+b} - alloc - hazardous
103  alloc ← alloc ∪ {m}
104  Allocated.write(alloc)
105  for all i ∈ used - hazardous do
106  TS[i].reset()
107  used ← used ∩ hazardous
108  Items[m].write(x)
109  InsertDone.dWrite()
110  return OK
111  else if TakeDone.dRead() = false then
112  return FULL
113  Take() by P_i, for i ∈ {1,...,n}:
114  InsertDone.dRead()
115  repeat
116  allocated ← Allocated.read()
117  for all a ∈ allocated do
118  Hazards[i].write(a)
119  x ← Items[a].read()
120  if x ≠ ⊥ then
121  if TS[a].t&s() = 0 then
122  Hazards[i].write(⊥)
123  TakeDone.dWrite()
124  return x
125  Hazards[i].write(⊥)
126  if InsertDone.dRead() = false then
127  TakeDone.dWrite()
128  return EMPTY
```

Figure 4 A lock-free, strongly-linearizable b-bounded bag with one producer and n consumers.

Linearizability. Consider any execution consisting of operations on this data structure. We linearize the operations as follows:

- An `Insert` operation that obtains `false` when it reads `TakeDone` on Line 111 is linearized at this read. It returns FULL.
- Consider an `Insert(x)` operation, *ins*, that allocated location a to `m` on Line 102 and wrote x to `Items[a]` on Line 108. Suppose that, while it is poised to write to `InsertDone` on Line 109, some `Take` operation, *tk*, performs a successful `t&s()` on `TS[a]` and then some (possibly different) `Take` operation writes to `TakeDone`. In this case, *ins* is linearized immediately before *tk*, *ins* returns OK, and we say that *ins* and *tk* are *coupled*.
  In the full version of the paper [10], we prove that at most one successful `t&s()` can be performed on `TS[a]` after *ins* writes to `Items[a]` and before it writes to `InsertDone`. Hence, an `Insert` operation is coupled with at most one `Take` operation.
- Suppose that an `Insert` operation, *ins*, writes to `InsertDone` on Line 109, but has not already been linearized as a coupled operation. Then, *ins* is linearized when it performs this write and returns OK, and we say that *ins* is *uncoupled*.
- Consider a `Take` operation, *tk*, that performs a successful `t&s()` on `TS[a]` on Line 121. It is linearized at the first among the following events to occur after the successful `t&s()` by *tk*: a write to `TakeDone` by any `Take` operation on Line 123 or Line 127, a read of 1 from `TS[a]` on Line 96, and a write to `InsertDone` on Line 109 by an uncoupled `Insert` that wrote to `Items[a′]` for some $a' \neq a$. Multiple successful `Take` operations linearized at the same step are ordered arbitrarily and before any other operations linearized at this step, with one exception: a coupled `Insert` is linearized right before its coupled `Take` operation. In all cases, *tk* returns the value it last read on Line 119.
- Consider a `Take` operation, *tk*, that obtains `false` when it reads `InsertDone` on Line 126. If no uncoupled `Insert` operation writes to `InsertDone` while *tk* is poised to write to `TakeDone`, then *tk* is linearized at its write to `TakeDone` (after any other operations linearized at this step). Otherwise, *tk* is linearized at the first such write to `InsertDone`, before the `Insert` and after any successful `Take` operations linearized at this step. The ordering among unsuccessful `Take` operations linearized at this write is arbitrary. In both cases, *tk* returns EMPTY.

To make the linearization points clearer, we also list the types of steps at which operations are linearized. For each, we specify the operations that can be linearized there. If multiple operations can be linearized at the same step, we specify the ordering of these operations.

- An `Insert` reads 1 from `TS[m]` on Line 96:
  - A **successful `Take`** that previously performed a successful `t&s` on `TS[m]`.
- A **successful uncoupled `Insert`**, *ins*, which wrote to `Items[m]`, writes to `InsertDone` on Line 109:
  - **Successful `Take`s** that previously performed a successful `t&s` on `TS[m′]` for $m' \neq m$, arbitrarily ordered.
  - **Unsuccessful `Take`s**, arbitrarily ordered.
  - *ins*.
- An **unsuccessful `Insert`**, *ins*, obtains false from `TakeDone.dRead()` on Line 111:
  - *ins*.
- A **successful `Take`** writes to `TakeDone` on Line 123:
  - **Successful `Take`s**, arbitrarily ordered.
    - **Successful coupled `Insert`s**, each immediately before its coupled `Take`.

- An **unsuccessful `Take`** *tk* writes to `TakeDone` on Line 127:
  - **Successful `Takes`**, arbitrarily ordered.
    - **Successful coupled `Inserts`**, each immediately before its coupled `Take`.
  - *tk*.

Next, we provide some intuition explaining our choice of linearization points, as well as why unsuccessful `Take` operations write to `TakeDone` on Line 127 before returning EMPTY.

We pick the linearization point for an `Insert` operation as we do in Section 6: a successful uncoupled Insert is linearized at its write to `InsertDone`, a successful coupled `Insert` operation is linearized right before its coupled `Take` operation, and an unsuccessful `Insert` operation is linearized when it obtains false from `TakeDone.dRead()` on Line 111.

We want to linearize a successful `Take` operation at its write to `TakeDone`, which is intended to notify the producer that the bag is not full, similarly to how a successful uncoupled `Insert` operation notifies the consumers that the bag is not empty. However, in the following four situations, we linearize a successful `Take` operation earlier than its write to `TakeDone`.

Suppose a successful `Take` operation, *tk*, has performed a successful `t&s` operation on Line 121, but has not yet been linearized when another successful `Take` operation writes to `TakeDone`. Then *tk* is linearized at this write. Linearizing *tk* at the earliest possible write to `TakeDone` in this case is done to simplify the linearization.

An unsuccessful `Take` operation, *tk*, cannot be linearized when it obtains false from `InsertDone.dRead()`, because there might be successful `Take` operations that remove elements from the bag, but are not yet linearized. These operations must be linearized before *tk*. To ensure this, *tk* writes to `TakeDone` before returning EMPTY. This linearizes every successful `Take` operation that has performed a successful `t&s` operation on Line 121, but has not been linearized before the write to `TakeDone` by *tk*. In this case, *tk* is linearized at this write, after all such successful `Take` operations.

A coupled `Take` operation witnesses that an `Insert` operation has happened and causes it to be linearized. Similarly, an `Insert` operation, *ins*, that witnesses a successful `Take` operation causes it to be linearized: If *ins* reads 1 from `TS[a]` on Line 96, but the `Take` that last performed a successful `TS[a].t&s()` is not yet linearized, then the `Take` is linearized at this `read`, which is before *ins* removes *a* from `alloc` on Line 98. Then, when *ins* sees that `|alloc|` $< b$ on Line 100, the bag is not full and it may insert a new element.

Suppose an unsuccessful `Take` operation, *tk*, got false from `InsertDone.dRead()`, but has not yet written to `TakeDone` when an uncoupled `Insert` operation, *ins*, writes to `InsertDone` on Line 109. Then *tk* must be linearized while the bag is still empty. Hence *tk* is linearized before *ins*, at this `dWrite`. Each successful `Take` operation that performed a successful `t&s` on Line 121, but has not yet been linearized, is also linearized at this step, before *tk*, to ensure the bag is empty when *tk* is linearized. There is one exception: If *ins* wrote to `Items[a]` on Line 108, a `Take` operation that performed a successful `TS[a].t&s()` at the same location after this write should be linearized after *ins*. A `Take` operation that performed a successful `TS[a].t&s()` at the same location before this write is guaranteed to be linearized before *ins*. This is because it is linearized at or before the producer last read 1 from `TS[a]` on Line 96, which occurs before the producer removes location *a* from `alloc` on Line 98, which, in turn, must occur before *ins* is allocated location *a* on Line 102.

The proof that Figure 4 is a lock-free, strongly-linearizable implementation of a b-bounded bag appears in the full version of the paper [10].

8 Discussion

This paper explores strongly-linearizable implementations of bags from interfering objects. We presented the first lock-free, strongly-linearizable implementation of a bag from interfering objects, which is, interestingly, also a linearizable implementation of a queue, but not a strongly-linearizable implementation. We also presented several implementations of bounded bags with a single producer from interfering objects: a wait-free, linearizable implementation of a 1-bounded bag, a lock-free, strongly-linearizable implementation of a 1-bounded bag, and a lock-free, strongly-linearizable implementation of a b-bounded bag for any value of b.

A direct extension of this work is investigating whether it is possible to extend our bounded bag implementations to support two producers or multiple producers but only one or two consumers. Other open questions are whether there are wait-free, strongly-linearizable implementations of ABA-detecting registers and bags from interfering objects. It would also be interesting to construct a lock-free, strongly-linearizable implementation of a bag from interfering objects, such that, at every point in the execution, the number of interfering objects used is bounded above by a function of the number of processes and the number of elements the bag contains. More broadly, exploring strongly-linearizable implementations of objects beyond bags using interfering objects is a compelling direction for future work.

References

1 Yehuda Afek, Eli Gafni, and Adam Morrison. Common2 extended to stacks and unbounded concurrency. *Distributed Computing*, 4(20):239–252, 2007. doi:10.1007/s00446-007-0023-3.

2 Zahra Aghazadeh and Philipp Woelfel. On the time and space complexity of aba prevention and detection. In *PODC*, pages 193–202, 2015. doi:10.1145/2767386.2767403.

3 Hagit Attiya, Armando Castañeda, and Constantin Enea. Strong linearizability using primitives with consensus number 2. In *PODC*, pages 432–442, 2024. doi:10.1145/3662158.3662790.

4 Hagit Attiya, Armando Castañeda, and Constantin Enea. Strong linearizability using primitives with consensus number 2. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.13618.

5 Hagit Attiya, Armando Castañeda, and Danny Hendler. Nontrivial and universal helping for wait-free queues and stacks. *J. Parallel Distributed Comput.*, 121:1–14, 2018. doi:10.1016/j.jpdc.2018.06.004.

6 Hagit Attiya and Constantin Enea. Putting strong linearizability in context: Preserving hyperproperties in programs that use concurrent objects. In *DISC*, pages 2;1–2:17, 2019. doi:10.4230/LIPIcs.DISC.2019.2.

7 Elizabeth Borowsky, Eli Gafni, and Yehuda Afek. Consensus power makes (some) sense! In *PODC*, pages 363–372, 1994. doi:10.1145/197917.198126.

8 Matei David. A single-enqueuer wait-free queue implementation. In *DISC*, volume 3274 of *Lecture Notes in Computer Science*, pages 132–143, 2004. doi:10.1007/978-3-540-30186-8_10.

9 Oksana Denysyuk and Philipp Woelfel. Wait-freedom is harder than lock-freedom under strong linearizability. In *DISC*, volume 9363 of *Lecture Notes in Computer Science*, pages 60–74, 2015. doi:10.1007/978-3-662-48653-5_5.

10 Faith Ellen and Gal Sela. Strong linearizability without compare&swap: The case of bags. *arXiv preprint*, 2025. arXiv:2411.19365.

11 Wojciech Golab, Lisa Higham, and Philipp Woelfel. Linearizable implementations do not suffice for randomized distributed computation. In *STOC*, pages 373–382, 2011. doi:10.1145/1993636.1993687.

12 Maryam Helmi, Lisa Higham, and Philipp Woelfel. Strongly linearizable implementations: possibilities and impossibilities. In *PODC*, pages 385–394, 2012. doi:10.1145/2332432.2332508.

13 Maurice Herlihy. Wait-free synchronization. *TOPLAS*, 13(1):124–149, 1991. doi:10.1145/114005.102808.

14 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *TOPLAS*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

15 Steven Munsu Hwang and Philipp Woelfel. Strongly linearizable linked list and queue. In *OPODIS*, pages 28–1, 2021. doi:10.4230/LIPIcs.OPODIS.2021.28.

16 Zongpeng Li. Non-blocking implementations of queues in asynchronous distributed shared-memory systems. Master's thesis, University of Toronto, 2001. Department of Computer Science.

17 Maged M. Michael. Hazard pointers: safe memory reclamation for lock-free objects. *TPDS*, 15(6):491–504, 2004. doi:10.1109/TPDS.2004.8.

18 Maged M. Michael and Michael L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *PODC*, pages 267–275, 1996. doi:10.1145/248052.248106.

19 Maged M. Michael and Michael L. Scott. Nonblocking algorithms and preemption-safe locking on multiprogrammed shared memory multiprocessors. *J. Parallel Distributed Comput.*, 51(1):1–26, 1998. doi:10.1006/jpdc.1998.1446.

20 Sean Ovens and Philipp Woelfel. Strongly linearizable implementations of snapshots and other types. In *PODC*, pages 197–206, 2019. doi:10.1145/3293611.3331632.

21 R Kent Treiber. Systems programming: Coping with parallelism. Technical Report RJ 5118, IBM Almaden Research Center, 1986.

A An Implementation of a Bag that is not Strongly-Linearizable

Consider the following lock-free, linearizable implementation of a queue [16]. We show that it is not a strongly-linearizable implementation of a bag.

```
129 Shared variables:
130    Items[1..]: an infinite array of registers, each initialized to ⊥
131    TS[1..]: an infinite array of test&set objects, each initialized to 0
132    Max: a readable fetch&increment object, initialized to 1
133 Insert(x):
134    max ← Max.f&i()
135    Items[max].write(x)
136 Take():
137    taken_old ← 0
138    max_old ← 0
139    repeat
140       taken_new ← 0
141       max_new ← Max.read() - 1
142       for i ← 1 to max_new do
143          x ← Items[i].read()
144          if x ≠ ⊥ then
145             if TS[i].t&s() = 0 then return x
146             taken_new ← taken_new + 1
147       if (taken_new = taken_old) and (max_new = max_old) then return EMPTY
148       taken_old ← taken_new
149       max_old ← max_new
```

Figure 5 An implementation of a queue that is not a strongly-linearizable bag.

Let ins_1 be a call of `Insert(1)`, let ins_2 be a call of `Insert(2)`, and let tk be a call of `Take()`. Consider the following execution α of their algorithm:

- ins_1 performs its f&i of `Max` on Line 134.
- ins_2 performs its f&i of `Max` on Line 134.

- tk performs Line 137, Line 138, and does an entire iteration of the repeat loop starting on Line 139, in which it sets *max_new* to 2 on Line 141 and sets *max_old* to 2 on line Line 149.
- tk starts a second iteration of the repeat loop, in which it sets *max_new* to 2 on Line 141 and reads $\perp$ from `Items[1]` on Line 143. However, it does not read `Items[2]` on Line 143.
- ins_1 writes 1 to `Items[1]` on Line 135 and returns.

Since ins_1 completes in α, it must be linearized in α.

Let α_1 be the continuation of α in which tk runs by itself to completion. It returns EMPTY as it encounters $\perp$ in every entry of `Items` it reads. Since tk returns EMPTY in $\alpha \cdot \alpha_1$, it must be linearized before ins_1. Hence, tk must be linearized in α.

Let α_2 be another continuation of α, in which ins_2 writes 2 to `Items[2]` on Line 135 and returns, and then tk runs by itself to completion. From the code, tk reads 2 from `Items[2]` and returns 2. This implies that ins_2 must be linearized before tk in $\alpha \cdot \alpha_2$ and, hence, in α. Therefore tk returns 2 in $\alpha \cdot \alpha_1$, which is a contradiction.

This shows that the algorithm in Figure 5 is not a strongly-linearizable implementation of a bag for any choice of linearization points. In particular, there is a problem with the linearization points mentioned in [4]:

- Each `Insert` operation is linearized when it performs Line 135.
- Each successful `Take` operation is linearized when it performs a successful `test&set` on Line 145.
- Each unsuccessful `Take` operation is linearized when it last reads `Max` on Line 141.

The issue is that, when an unsuccessful `Take` operation is linearized, the bag might not be empty. Let ins be a call of `Insert(1)` and let tk_1 and tk_2 be calls of `Take()`. Consider the following execution:

- ins performs its f&i of `Max` on Line 134.
- tk_1 performs Line 137, Line 138, and does an entire iteration of the repeat loop starting on Line 139 in which it sets *max_new* to 1 on Line 141 and sets *max_old* to 1 on line Line 149.
- ins writes 1 to `Items[1]` on Line 135 and returns.
- tk_1 starts a second iteration of the repeat loop, in which it sets *max_new* to 1 on Line 141.
- tk_2 performs a `Take` operation, returning the value 1 when it first performs Line 145.
- tk_1 completes its `Take` operation, returning EMPTY when it returns on Line 147.

Note that, when tk_1 last performs Line 141, where it is linearized, the bag contains the element 1.

B A Proof that Figure 1 is not a Strongly-Linearizable Implementation of a Queue

We show that our strongly-linearizable implementation of a bag is not a strongly-linearizable implementation of a queue.

Let ins_1 be a call of `Insert(1)`, let ins_2 be a call of `Insert(2)`, and let tk, tk_1, and tk_2 be calls of `Take()`. Consider the following execution α of the algorithm in Figure 1:

- ins_1 performs its f&i of `Allocated` on Line 7 and is allocated location 1.
- ins_2 performs its f&i of `Allocated` on Line 7 and is allocated location 2.
- tk reads 2 from `Allocated` on Line 13 and then reads $\perp$ from `Items[1]` on Line 15.
- ins_1 and ins_2 run to completion.

Since ins_1 and ins_2 complete in α, they must be linearized in α.

Let α_1 be the continuation of α in which tk runs by itself to completion. It reads 2 from `Items[2]` on Line 15 during its second iteration of the for loop and returns 2. To satisfy the semantics of a queue, ins_2 must be linearized before ins_1 in $\alpha \cdot \alpha_1$ and, hence, in α.

Let α_2 be another continuation of α, in which tk_1 runs by itself to completion and then tk_2 runs by itself to completion. From the code, tk_1 reads 1 from `Items[1]` and returns 1, and tk_2 reads 2 from `Items[2]` and returns 2. To satisfy the semantics of a queue, tk_1 must be linearized before tk_2 in $\alpha \cdot \alpha_2$ and, hence, in α. This is a contradiction.

C A Proof that Figure 2 is not a Strongly-Linearizable Implementation of a 1-Bounded Bag

We show that our wait-free implementation of a 1-bounded bag with a single producer is not strongly-linearizable.

Let ins_i be a call of `Insert(`i`)` and tk_i a call of `Take()`, for $1 \leq i \leq 3$. Consider the following execution α of the algorithm in Figure 2:

- ins_1 starts running, and it is allocated location 1 on Line 32. It then runs to completion.
- tk_1 runs and returns 1.
- tk_2 reads 1 from `Allocated` on Line 40.
- ins_2 starts running. It writes $\perp$ to `Items[1]` on Line 29. It is allocated location 2 on Line 32. It then runs to completion.

Let α_1 be the continuation of α in which tk_2 runs by itself to completion. It returns EMPTY as it encounters $\perp$ in every entry of `Items` it reads including `Items[1]`. Since tk_2 returns EMPTY in $\alpha \cdot \alpha_1$, it must be linearized before ins_2. Hence, tk_2 must be linearized in α, with return value EMPTY.

Let α_2 be another continuation of α, in which the following occur:

- tk_3 runs and returns 2.
- ins_3 starts running. It is allocated location 1 on Line 32. It continues to run, writes 3 to `Items[1]` and returns.
- tk_2 runs to completion.

From the code, tk_2 reads 3 from `Items[1]` and returns 3. Hence, tk_2 must be linearized after ins_3 in $\alpha \cdot \alpha_2$. This contradicts strong-linearizability.

Team Formation and Applications

Yuval Emek
Technion, Haifa, Israel

Shay Kutten[1]
Technion, Haifa, Israel
GTEP, Haifa, Israel

Ido Rafael
Technion, Haifa, Israel

Gadi Taubenfeld
Reichman University, Herzliya, Israel

Abstract

A novel long-lived distributed problem, called *Team Formation (TF)*, is introduced together with a message- and time-efficient randomized algorithm. The problem is defined over the asynchronous model with a complete communication graph, using bounded size messages, where a certain fraction of the nodes may experience a generalized, strictly stronger, version of initial failures. The goal of a TF algorithm is to assemble tokens injected by the environment, in a distributed manner, into *teams* of size σ, where σ is a parameter of the problem.

The usefulness of TF is demonstrated by using it to derive efficient algorithms for many distributed problems. Specifically, we show that various (one-shot as well as long-lived) distributed problems reduce to TF. This includes well-known (and extensively studied) distributed problems such as several versions of leader election and threshold detection. For example, we are the first to break the linear message complexity bound for asynchronous implicit leader election. We also improve the time complexity of message-optimal algorithms for asynchronous explicit leader election. Other distributed problems that reduce to TF are new ones, including matching players in online gaming platforms, a generalization of gathering, constructing a perfect matching in an induced subgraph of the complete graph, and more. To complement our positive contribution, we establish a tight lower bound on the message complexity of TF algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases asynchronous message-passing, complete communication graph, initial failures, leader election, matching

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.30

Related Version A full version of this extended abstract is available, containing details omitted here. *Full Version*: `https://arxiv.org/abs/2508.13084` [22]

Funding *Yuval Emek*: Partially supported by an Israel Science Foundation (ISF) grant 730/24 and by the Grand Technion Energy Program (GTEP).
Shay Kutten: Partially supported by ISF grant 1346/22; his research was carried out within the framework of the Grand Technion Energy Program (GTEP).

Acknowledgements We would like to thank Eviatar Procaccia for his help with the proof of Lemma 9.

1 Introduction

Consider the following three problems, defined over an asynchronous message-passing system with a complete communication graph, where a constant fraction of the nodes may be faulty: (P1) electing a leader among the non-faulty nodes; (P2) constructing a perfect matching in

[1] a part of the work was done while visiting Fraunhofer SIT

© Yuval Emek, Shay Kutten, Ido Rafael, and Gadi Taubenfeld;
licensed under Creative Commons License CC-BY 4.0

the subgraph induced by a subset of the non-faulty nodes, specified (distributively) by the input; (P3) assembling Dungeons & Dragons parties from characters that arrive over time at the non-faulty nodes so that each party includes 3 wizards, 2 paladins, 2 rogues, and 1 monk. On the face of it, problems (P1)–(P3) have little in common. In particular, (P1) and (P2) are one-shot problems, whereas (P3) is long-lived; (P1) requires global symmetry breaking, whereas in (P2) and (P3) the symmetry breaking is local in essence. Therefore, it may come as a surprise that all three problems reduce to a single new problem.

In this paper, we introduce the aforementioned new problem, called *team formation*, and develop an efficient algorithm for it. Following that, we show how our algorithm leads to algorithmic solutions for various other problems (including (P1)–(P3)), improving their state-of-the-art. For example, we are the first to break the linear message complexity bound for asynchronous implicit leader election, see Sec. A.1.1. We also improve the time complexity of message-optimal algorithms for asynchronous explicit leader election, see Sec. A.1.2. On the negative side, we establish lower bounds on the communication demands of TF algorithms, see the full version [22].

1.1 The Basic Setting and Problem Definition

Consider an *asynchronous* message-passing system that consists of n nodes (a.k.a. processors). The communication structure is assumed to be a complete undirected graph over the node set V so that every two nodes may exchange messages with each other. These messages carry $O(\log n)$ bits of information (cf. the CONGEST model [52]) and are delivered with a finite delay, where the delay of each message is determined (individually) by an adversary. Unless stated otherwise, it is assumed that the nodes are anonymous and that each node distinguishes between its $n-1$ neighbors by means of (locally) unique *port numbers* [6, 61, 31]. The adversary may cause some nodes to become faulty, preventing them from participating in the execution in accordance with a generalization of the *initial failures* model [18, 24, 60], as long as a sufficiently large fraction of the nodes remain non-faulty throughout the execution.[2] A precise definition of the computational model is presented in Sec. 3.1.

Team Formation. We introduce a long-lived distributed problem called *team formation (TF)*, defined over an integral *team size* parameter $2 \leq \sigma \leq n$.[3] In a TF instance, abstract *tokens* are injected into the nodes over time, where the adversary determines the timing and location of these token injections. Tokens held by a node $v \in V$ can be transported to a node $v' \in V$ over a message sent from v to v'. Each token remains in the system until it is deleted, as explained below.

The correctness criterion of the TF problem is captured by the following two conditions:

Safety: a token can be deleted from the system only as part of a *team* that consists of exactly σ tokens, all of which are held by the same node and deleted simultaneously; the operation of deleting a team of tokens is referred to as *forming a team*.

Liveness: if the system contains at least σ tokens, then a team must be formed in a finite time.

[2] The failure model considered in this paper is identical to the classic model of initial failures when restricted to one-shot problems. As explained in the sequel, the generalization comes into play only in the context of long-lived problems.

[3] The TF problem is well defined also for $\sigma > n$, however, the restriction to $\sigma \leq n$ simplifies the discussion and is consistent with the applications presented in Sec. 1.4.

Assume without loss of generality that the system contains at most $n^{O(1)}$ tokens at any given time.[4] The tokens are assumed to be indistinguishable and any number of tokens can be transported over a single message that simply encodes their number.

1.2 Our Contribution

Our main contribution regarding the TF problem is cast in Thm. 1 (for the formal statement, refer to the analysis sections provided in the full version [22]).

▶ **Theorem 1** (slightly informal). *For every constant $\epsilon > 0$, there exists a randomized algorithm that solves any n-node TF instance whp if at least ϵn nodes remain non-faulty indefinitely.*[5] *The algorithm sends $O(\sqrt{n \log n})$ messages per token in expectation and $O(\sqrt{n \log n} \cdot \log n)$ messages per token whp. Moreover, if the system contains at least σ tokens, then the algorithm is guaranteed to form a team in $O(\sigma + \log n)$ (asynchronous) time whp.*

We emphasize that the whp guarantees promised in Thm. 1 apply for any TF instance, regardless of the total number of tokens injected into the system that may be arbitrarily large (and not necessarily bounded as a function of n); in particular, the correctness probability remains (arbitrarily) close to 1 and does not deteriorate as the number of tokens increases. Moreover, the bounds on the number of messages per token hold already for "the first tokens", that is, even if the adversary decides to inject a few tokens in total, regardless of σ. It is also interesting to point out that our TF algorithm withstands a $(1 - \epsilon)$-fraction of faulty nodes for an arbitrarily small constant $\epsilon > 0$;[6] this is in contrast to many other problems (in message-passing with initial failures), including leader election and consensus, where a majority of faulty nodes leads to impossibility results. On the negative side, we establish the following theorem (refer to the full version [22] for the formal statement).

▶ **Theorem 2** (slightly informal). *For $2 \leq \sigma \leq n/2$, consider the simplified (one-shot) version of the TF problem, where the schedule is synchronous and it is guaranteed that exactly σ tokens are injected into the system, all at the beginning of the execution. Any algorithm that solves this problem whp must send a total of $\Omega(\max\{\sqrt{n \log n}, \sqrt{n\sigma}\})$ messages in expectation. This holds even if the messages are of unbounded size.*

1.3 Extra Features

Beyond the safety and liveness conditions presented in Sec. 1.1, it may be advantageous for TF algorithms to satisfy additional features. In this section, we introduce three such desirable features that turn out to be very useful (see Sec. 1.4), all three of them are readily satisfied by the TF algorithm promised in Thm. 1. Refer to Sec. C for further discussions of the qualities of these features.

The Forgetful Feature. In the scope of this feature, we distinguish between two types of coin tosses of a node $v \in V$: (1) the finitely many "factory coin tosses" generated by v upon the first activation event of v, before any other action of v and hence, could have been

[4] Conditioned on the assumption that $\sigma \leq n$, the nodes themselves should hold (all together) less than n^2 tokens since a node that holds at least σ tokens can perform team formation operations. Regarding the tokens in transit, the algorithm developed in the current paper is designed so that there are less than σ tokens in transit over any edge at any given time, so the total number of tokens in transit is $O(n^3)$.

[5] An event A occurs *with high probability (whp)* if $\mathbb{P}(A) \geq 1 - n^{-c}$ for any (arbitrarily large) constant c.

[6] Refer to Sec. 3.1 for a precise definition of the failure model adopted in the current paper, including the assumptions made on the faulty nodes.

generated when v was "manufactured"; (2) all other coin tosses of v, generated along the execution. In contrast to the latter, the "factory coin tosses" are considered to be hardwired into v's memory and, in particular, constitute part of v's initial state.

We say that a randomized algorithm satisfies the *forgetful feature* if it is guaranteed that every node resides in its initial state at any quiescent time (formally defined in Sec. 3.1). Thus, at quiescent times, the nodes may not record any information about their past events and actions. We view the notion of forgetful algorithms as a natural extension of the important notion of *memoryless* algorithms [57, 59] from deterministic to randomized algorithms and believe that it should be studied further, regardless of the TF problem.

The Trace-Tree Feature. A token τ injected into node $v \in V$ follows a tour in the communication graph from v to the node r at which τ is deleted (if at all) as part of a team formation event F. Somewhat informally, this tour forms a simple path π in the temporal graph that reflects the communication graph along the execution's time axis (a node $v' \in V$ may appear in π several times, but at different times). The collection of the paths π of all the tokens τ that participate in F form a temporal tree T rooted at r. A TF algorithm satisfies the *trace-tree feature* if the algorithm maintains a distributed data structure that supports broadcast and echo processes over T (it is the responsibility of "the user" to delete this data structure once it is no longer used). Since the basic description of our TF algorithm, as presented in Sec. 4, does not address this trace-tree feature, we provide the implementation details in [22].

The Accumulation Feature. Consider a TF instance and assume that the total number ℓ of tokens injected into the system satisfies $\ell \bmod \sigma = k > 0$. A TF algorithm satisfies the *accumulation feature* if it is guaranteed that the k tokens that remain in the system forever are eventually held by the same node.

1.4 Applications

We present several interesting problems that are reducible to TF. These include the classic leader election problem, for which we improve the state-of-the-art, as well as various problems that received less attention from the community so far (if at all). For each problem, we provide a sketchy description of the reduction to TF; further technical details as well as precise statements of the results obtained through these reductions are deferred to Sec. A.

Leader Election (LE). This is the fundamental one-shot problem of designating a single non-faulty node of a communication network as a leader [49, 7, 52]. We improve upon the previous results for both (1) the *implicit* version, where each node is required to know whether it is the elected leader; and (2) the *explicit* version, where, in addition to the above requirement, each non-leader knows which of its (internal) ports leads to the leader. Moreover, our algorithms are fault-tolerant (see Sec. 3.1), while the improved upon prior art [43, 44] assumes fault freedom.

Intuitively, the reduction works as follows: A logarithmic number of nodes are chosen probabilistically to serve as candidates, injecting a token into each one of them. The team size parameter is adjusted so that the number of injected tokens suffices for the formation of exactly one team whp. The node at which the team is formed is elected. In the explicit LE version, on top of the above, the elected leader notifies all other nodes.

We assume that there are at most $n(1/2 - \epsilon)$ faulty nodes for any constant $\epsilon > 0$. The run-time of our LE algorithms is $O(\log n)$ whp. Our *explicit* LE algorithm sends $O(n)$ messages in expectation and whp. This improves upon the time complexity of the best

previous algorithms with $O(n)$ messages for asynchronous explicit LE [43, 44]; their time complexity was $(\log^2 n)$ even though they assume fault-freedom. Our *implicit* LE algorithm sends $O(\sqrt{n \log n})$ messages in expectation and $O(\sqrt{n \log n} \cdot \log n)$ messages whp. This is the first algorithm with time complexity $o(n)$ for asynchronous LE (even for fault-free algorithms).

Vector Team Formation. Consider a (long-lived) generalization of the TF problem, where each token comes with a *color* from a certain color palette and a team should include a pre-specified number of tokens from each color. This problem, called *vector team formation (vTF)*, is formulated as follows: Instead of a scalar team size parameter $\sigma \in \mathbb{Z}_{>0}$, the vTF problem is defined over a *team size vector* $\vec{\sigma} \in \mathbb{Z}_{>0}^m$ whose dimension m corresponds to the size of the color palette; a team now consists of (exactly) $\vec{\sigma}(i)$ tokens of color i for each $i \in [m]$. As in the TF problem, the safety condition states that tokens can be deleted only as part of a team, whereas the liveness condition states that if the system contains at least $\vec{\sigma}(i)$ tokens of color i for each $i \in [m]$, then a team must be formed in a finite time. (The aforementioned Dungeons & Dragons example is a special case of vTF.)

To solve vTF, we shall invoke a separate copy of a TF algorithm for each color $i \in [m]$, setting $\sigma = \vec{\sigma}(i)$, and generate a "super-token" of color i whenever the i-th copy forms a team. The super-tokens can then be collected into a vTF team, exploiting the fact that the symmetry is now broken due to the different colors.

Agreement with Failures. The problem of (explicit) agreement with initial failures was defined in the seminal paper of [24] as a contrast to the setting of asynchronous crash failures proved to be impossible. A weaker version of the problem, known as *implicit* agreement, is defined in [8]. Informally, the latter problem requires that (1) at least one node must decide; (2) the decided value must be the same for all deciding nodes; and (3) the decided value must be the input of some node. This version of the agreement problem was addressed so far only in *synchronous* networks, considering both fault-free environments [8] and crash faults [40]. As implicit leader election is directly reducible to the aforementioned implicit leader election problem, we obtain an *asynchronous* implicit leader election algorithm that withstands $n(1/2 - \epsilon)$ faults in the initial failures model.

Online Gaming Platforms. This long-lived problem addresses an online gaming platform (see, e.g., [23]) in a "one-versus-one" game mode (two players compete against each other directly like in chess), where players arrive over time and should be matched with opponents. The reduction to TF works as follows: Each player is represented by a single token, which is injected into a relevant node, e.g., a nearby server, and the TF algorithm is invoked with $\sigma = 2$. When a team is formed, the trace-tree is used to establish a connection between the servers of the matched players.

Distributed Matching with Failures. In this one-shot problem, the nodes are equipped with unique IDs and an (even size) subset of the (non-faulty) nodes is marked as part of the input. The goal is to form a perfect matching in the subgraph induced by the marked nodes so that each node knows the ID of the node it is matched to. To reduce this problem to TF, intuitively, each marked node generates a token and the TF algorithm is invoked with $\sigma = 2$; the trace-tree is employed to allow the matched nodes to exchange IDs.

Robot Team Gathering. The one-shot *gathering* problem requires robots, initially positioned arbitrarily, to meet at a single point within a finite time, where the meeting point is not fixed initially [5, 48, 51, 25, 58]. We propose and solve a new problem called *robot team gathering* which is a generalization of the gathering problem. Robots are required to form teams of a given (known) size, and each team must meet at a single point within a finite time, where the meeting points are not fixed initially. In the reduction of robot team gathering (in a complete graph) to TF, each robot is represented as a token, and the TF algorithm is used to form teams.[7]

Distributed Trigger Counting. Consider a network of interconnected devices where the devices count triggers from an external source. In the one-shot *distributed trigger counting (DTC)* problem, the algorithm is required to raise an alarm when the total number of triggers counted by all the devices reaches a predefined threshold [17, 20, 37]. Intuitively, to solve DTC, each trigger generates a token, and the TF algorithm is invoked after setting the team size parameter to the threshold value.

Team Forming with Associated Information. The application is for the case where a token τ arrives with some associated information $\mathrm{AS}(\tau)$. It is required that $\mathrm{AS}(\tau)$ be available at the node where the team, that includes τ, is formed at the time of the formation. For example, $\mathrm{AS}(\tau)$ may be a piece of a secret, and σ pieces are needed and are enough to recover the secret. We further assume that the information associated with any two tokens that exist in the network simultaneously is different. This means that the size of $\mathrm{AS}(\tau)$ may be large, so only a constant number of such pieces may fit in one message (otherwise, the implementation may be straightforward even if the message size is $O(\log n)$). The reduction to (the standard version of) TF is slightly more technical, see [22].

1.5 Paper's Outline

In Sec. 2, we present the main technical challenges that arise in the study of the TF problem. Sec. 3 introduces the model, initial failures generalization, complexity measures, and basic definitions. Sec. 4 provides a high-level description of the algorithm, structured into two layers and the interface between them. Sec. 5 presents the main analysis of the algorithm, establishing safety and liveness and analyzing the message load. Sec. A contains additional technical details (omitted from Sec. 1.4) for LE as an application of our TF algorithm. Further related work is surveyed in Sec. B. Sec. C offers, a discussion on the usefulness of the TF problem, the results presented, and future directions.

Due to space considerations, some details are omitted from this extended abstract and deferred to the full version [22]. This includes technical details of the algorithm's lower layer, an analysis of the algorithm's reaction time, our lower bound (stated in Thm. 2), further details on the trace-tree mechanism mentioned in Sec. 1.4, additional applications of the TF problem beyond LE (see Sec. 1.4), and various figures, tables, and pseudocodes.

[7] It is not difficult to have the robots (each playing a token) to simulate the algorithms of the nodes, using whiteboards [19, 26] at the nodes.

2 Technical Challenges

Before we move on to presenting our solution for the TF problem, the reader may wonder about the following simple scheme: Elect a leader v (once) and following that, transport all injected tokens to v so that v handles all team formation operations. We argue that this scheme suffers from various severe shortcomings: (1) under our failure model (formally presented in Sec. 3.1), the leader v may become faulty at any quiescent time, causing all subsequently injected tokens to get lost; (2) since our model does not include "spontaneous wake-ups", it is not clear which nodes participate in the leader election stage; (3) leader election is impossible if we allow a $(1-\epsilon)$-fraction of the nodes to be faulty; and (4) the scheme does not satisfy the forgetful feature (as defined in Sec. 1.3).

The aforementioned shortcomings highlight some of the technical challenges we had to overcome when devising our TF algorithm. To discuss those, assume for now that the team size parameter σ is a large constant and consider a token τ injected into some node v. The first task faced by v is to look for other (nodes that hold) tokens – if there are at least $\sigma - 1$ of those, then a team should be formed. Doing so in a communication efficient manner is a challenge, especially under an asynchronous scheduler when a large fraction of the nodes may be faulty (and thus, never respond to incoming messages). We tackle this challenge by constructing a "probabilistic quorum system" with quorums of size $O(\sqrt{n \log n})$ (using the "factory coin tosses"), thus ensuring that any two token holding nodes have at least one non-faulty node in the intersection of their quorums.

Once node v has identified other token holding nodes, the next task is to determine who transports the tokens to whom, raising a symmetry breaking challenge: a naive approach may cause v to transport its token to v' while v' transports its own token to v'' and so on. To prevent long token transportation chains, we adopt a "star-shaped" transportation pattern: the token holding nodes run consecutive phases and in each phase, assume a center or an arm role at random; each arm node v_a then tries to transport its token to a center node v_c that is "ready to accept" v_a's token. For this idea to succeed, one has to lower-bound the probability for an arm node to identify an appropriate center node in each phase. However, as we cannot synchronize between the phases of different nodes, ensuring such a probability lower bound turns out to be eminently challenging. We resolve this challenge by incorporating a "selective waiting mechanism" inspired by a technique introduced in [9] (for the entirely different task of constructing a maximal independent set in a general graph); refer to Sec. 4 for details.

3 Preliminaries

3.1 The Computational Model

Fix some randomized TF algorithm `Alg`. Recall our assumption that the adversary assigns a finite delay to each message of `Alg`. The only constraint we impose on the adversary in this regard is that the messages sent from a node $v \in V$ to a node $v' \in V$ are delivered in FIFO order. For the sake of the runtime analysis (more on that later), we scale the time axis so that the message delays are up-bounded by 1 time unit; we emphasize that the nodes themselves have no notion of time.

Event Driven Executions. The execution of `Alg` advances through the continuous time axis $\mathbb{R}_{>0}$ in an event driven fashion so that a node $v \in V$ is activated at time $t > 0$ if (and only if) one of the following two *activation events* occurs at time t: (1) a message is delivered to v; or (2) one or more tokens are injected into v. When activated, the actions of v under

`Alg` include (I) reading the new incoming message (if any); (II) local computation that may involve (private) random coin tosses; (III) any number of team formation operations; and (IV) sending messages to (a subset of) v's neighbors.[8] It is assumed that actions (I)–(IV) are performed atomically (i.e., they take zero time). We emphasize that the nodes never act unless activation events (1) or (2) occur (in particular, in the scope of the TF problem, there are no "spontaneous wake-ups").

Assume without loss of generality that the activation events are isolated so that no two of them occur at the same time; the times at which these activation events occur are referred to as *activation times*. Time $t > 0$ is regarded as *quiescent* if (1) t is not an activation time; (2) there are no messages in transit at time t; and (3) the system contains 0 tokens at time t.

For a time $t > 0$, let $t- = t - dt$ and $t+ = t + dt$ be the times immediately before and immediately after t, respectively, where dt is chosen so that the time interval $[t-, t+]$ contains at most one activation event. Given an object `obj`, let $\texttt{obj}^t$ denote the state of `obj` at time t, adhering to the convention that $\texttt{obj}^t = \texttt{obj}^{t-}$ for all $t > 0$;[9] in particular, if t is an activation time and the state of `obj` changes at time t, then $\texttt{obj}^t$ is taken to be the state before the change. Notice that our convention implies that instantaneous changes to `obj`, that are done as part of the local computation and do not persist beyond time t, are not reflected in $\texttt{obj}^t$.

Fault Tolerance and Adversarial Policies. We adopt a generalization of the *initial failures* model [18, 24, 60]. Under the initial failures model, the adversary selects a node subset $F \subset V$ at time 0 (i.e., right before the execution commences) and turns the status of all the nodes in F to *faulty*, thus preventing them from participating in the execution. In particular, faulty nodes do not send messages and in the context of the TF problem, no tokens are injected into faulty nodes. A message μ delivered to a faulty node is lost (i.e., it is ignored and does not trigger an activation event). If tokens are transported over a lost message μ, then these tokens are assumed to remain in "limbo" forever, which may prevent the algorithm from satisfying the liveness condition; it is the responsibility of the algorithm designer to avoid such scenarios.

In the (new) generalized version of the initial failures model, considered in the current paper, the adversary may *toggle* the status of a node $v \in F$ from faulty to non-faulty and vice versa, so that v may participate in the execution in certain time intervals and not participate in others. The decisions of the adversary in this regard are subject to the following constraint: the faulty/non-faulty status of v may be toggled only at quiescent times. For a precise description of the failure model, if v becomes faulty at time $t \geq 0$ and remains faulty until time $t' > t$, at which it becomes non-faulty, then the memory image of v at time t' is assumed to be identical to that of time t. To avoid confusion, we subsequently refer to the nodes in F as *fragile* (regardless of their faulty/non-faulty status at a specific time of the execution), emphasizing that the non-fragile nodes remain non-faulty throughout the execution.

Notice that due to the asynchronous nature of the system, it may be impossible for a non-faulty node to distinguish its faulty neighbors from the "slow responding" ones. Moreover, the fragile nodes do not know that they are fragile and when a fragile node becomes non-faulty, it runs the same algorithm as the non-fragile nodes. The generalization of the classic initial failures model is different from the crash failures model, where a node may fail at any time. As pointed out in [24], considering initial failures is known to open the gate for asynchronous algorithms that are impossible under crash failures.

---

[8] We allow v to send multiple messages to the same neighbor (in practice, those can be piggybacked).
[9] Objects in this regard include local variables as well as global objects defined for the analysis.

With this failure model, the adversary has the following roles: (1) At time 0, the adversary determines the subset $F \subset V$ of fragile nodes and (without loss of generality) turns their status to faulty. (2) For time $t = 0$ and for every activation time $t > 0$, at time $t+$, the adversary determines the next activation event (if any) including its type (message delivery or token injection), location, and time $t' > t$. (3) For every quiescent time $t > 0$ and for every node $v \in F$, the adversary determines (at time t) if the faulty/non-faulty status of v is toggled at time t (notice that this is *not* an activation event). We emphasize that tokens can be injected into node v at time t only if v is non-faulty at time t; tokens are never injected into faulty nodes.

The adversary is *adaptive*, namely, its decisions at time t may be based on the nodes' actions before time t (however, without knowing the nodes' future coin tosses). Consequently, the execution of `Alg` can be represented as an extensive form game with incomplete information [30], where the adversary is the only strategic player; that is, a rooted tree whose (internal) vertices are partitioned into a set of adversarial moves, in which the adversary makes its decisions, and a set of chance moves, each corresponding to the coin tosses of an activated node in some activation event. Terminology-wise, a mechanism that makes the decision in each adversarial move of the aforementioned extensive form game is referred to as an *adversarial policy*. Notice that by fixing an adversarial policy, `Alg`'s execution becomes a random variable, fully determined by the nodes' coin tosses.

Performance Measures. The quality of `Alg` is evaluated by means of two performance measures. First and foremost, we wish to bound the number of messages that `Alg` sends per token in expectation and whp. Care is needed in this regard since the adaptive nature of the adversary implies that the number of tokens injected into the system may depend, by itself, on the nodes' coin tosses. Thus, the number of messages per token is, in general, the ratio of two random variables that may exhibit complex dependencies, making it difficult to reason about. To resolve this difficulty, we formulate our main performance measure through the notion of *message load*, defined as follows.

For $\ell \in \mathbb{Z}_{>0}$, let $\mathcal{A}_\ell$ be the family of adversarial policies that inject at most ℓ tokens into the system (throughout the execution). We say that `Alg`'s *message load* is M in expectation (resp., whp) if M is the smallest real such that for every $\ell \in \mathbb{Z}_{>0}$ and for every adversarial policy in $\mathcal{A}_\ell$, it is guaranteed that `Alg` sends at most $M \cdot \ell$ messages in expectation (resp., whp).[10]

We are also interested in the time it takes for `Alg` to form a team once sufficiently many tokens are present. To this end, we say that `Alg`'s *reaction time* is R if R is the smallest real such that for every time $t > 0$, if the system contains at least σ tokens at time t, then a team is formed by time $t + R$ whp.

3.2 The Primary-Utility Graph

The algorithms developed in this paper are designed so that each node $v \in V$ simulates a virtual *primary node* $p(v)$ and a virtual *utility node* $u(v)$; let $P = \{p(v) : v \in V\}$ and $U = \{u(v) : v \in V\}$ denote the sets of primary and utility nodes, respectively. Although the primary node $p(v)$ and utility node $u(v)$ are simulated by the same "physical" node $v \in V$,

[10] We note that the message load provides a bound on the number of messages sent (per token) in "finite prefixes" of the execution under *any* adversarial policy, including those that do not belong to $\bigcup_{\ell>0} \mathcal{A}_\ell$.

it is convenient to address them as standalone computational entities that participate in the execution by receiving and sending messages independently of v. To this end, tokens injected into v are regarded as injected into the primary node $p(v)$.

In terms of the failure model, the primary node $p(v)$ and utility node $u(v)$ are regarded as *fragile* if the corresponding "physical" node $v \in V$ is fragile. Let $F_P = \{p(v) \in P : v \in F\}$ and $F_U = \{u(v) \in U : v \in F\}$ denote the sets of fragile primary and utility nodes, respectively. The faulty/non-faulty status of the fragile primary node $p(v) \in F_P$ and fragile utility node $u(v) \in F_U$ is assumed to be inherited from that of the corresponding fragile "physical" node $v \in F$.[11]

A key design feature of our algorithms is that the entire message exchange is confined to an overlay bipartite graph $G_{PU} = (P, U, E_{PU})$, referred to as the *primary-utility graph*. The edge set E_{PU} of the primary-utility graph is constructed randomly by selecting each node pair $(p, u) \in P \times U$ to be included in E_{PU} independently with probability $c \cdot \sqrt{\frac{\log n}{n}}$, where $c > 0$ is a constant to be derived from the analysis;[12] this random selection is made by each primary node p upon its first activation event and remains fixed throughout the execution. Notation-wise, for a primary node $p \in P$ and a utility node $u \in U$, let $U(p) = \{u' \in U : (p, u') \in E_{PU}\}$ and $P(u) = \{p' \in P : (p', u) \in E_{PU}\}$. The key probabilistic properties of the primary-utility graph are cast in the following lemma (proof deferred to the full version [22]); in the remainder of this paper, we condition on the event that these properties are satisfied.

▶ **Lemma 1.** *The primary-utility graph $G_{PU} = (P, U, E_{PU})$ satisfies the following two properties whp:*
(1) $(U(p) \cap U(p')) - F_U \neq \emptyset$ *for every $p, p' \in P$; and*
(2) $|U(p)|, |P(u)| \leq O(\sqrt{n \log n})$ *for every $p \in P$ and $u \in U$.*

4 The Algorithm

In this section, we present our TF algorithm, referred to as $\texttt{Alg}_{\text{TF}}$. The algorithm attempts to (locally) gather tokens, that may be distributed over multiple primary nodes, into a single primary node and perform team formation(s) if the number of gathered tokens is large enough.

Under $\texttt{Alg}_{\text{TF}}$, each primary node $p \in P$ maintains the local variable $p.\texttt{tok} \in \mathbb{Z}_{\geq 0}$ that counts the number of tokens held by p. Recalling that $p.\texttt{tok}^t$ is the value of $p.\texttt{tok}$ at time $t > 0$, let $\mathcal{B}^t = \{p \in P : p.\texttt{tok}^t > 0\}$, referring to the nodes in $\mathcal{B}^t$ as *busy*. The algorithm is designed so that a utility node $u \in U$ may hold tokens only instantaneously, when the tokens are transported, through u, between two primary nodes in $P(u)$.

Fix some $p \in P$ and $t > 0$ and suppose that $p \in \mathcal{B}^t$ and that t is an activation time of p. If $p.\texttt{tok}^t < \sigma$, then p may decide, at time t, to transport the tokens it holds by sending a message μ, that carries these tokens, to another primary node $p' \in P$, through a utility node $u \in U(p) \cap U(p')$; the algorithm is designed so that this may happen only if p' is busy at time t and remains busy (at least) until message μ is delivered to p'. If $p.\texttt{tok}^t \geq \sigma$, then p may decide, at time t, to eliminate some of the tokens it holds, in which case, p performs $\lfloor p.\texttt{tok}^t/\sigma \rfloor$ team formations (i.e., as many team formations as possible). We emphasize that p will never transport tokens as long as $p.\texttt{tok} \geq \sigma$ and that if p does transport the $(p.\texttt{tok} < \sigma)$ tokens it holds, then all $p.\texttt{tok}$ tokens are transported together and p becomes non-busy.

[11] This assumption is made only for the sake of simplicity; as far as our algorithms and analyses go, the faulty/non-faulty status of $p(v)$ and $u(v)$ can be decoupled.
[12] With a slight abuse of notation, we often use p (resp., u) as a placeholder for a general primary (resp., utility) node.

If p performs team formation(s) at time t and $p.\texttt{tok}^t \bmod \sigma = k > 0$, then p is left with k "remainder tokens". To simplify the presentation, we treat these k "remainder tokens" as if they are injected into p, as fresh tokens, soon after time t (and strictly before the next "original" activation event); refer to the injections of such "remainder tokens" as *fake injections*. In particular, we assume that if p performs team formation(s) at time t, then $p.\texttt{tok}^{t+} = 0$, hence $p \notin \mathcal{B}^{t+}$. Put differently, as long as $p \in \mathcal{B}$, the variable $p.\texttt{tok}$ is a non-decreasing function of t.[13]

Two-Layer Structure. To simplify the algorithm's presentation, we divide it, logically, into two layers: a lower *channel layer* and an upper *principal layer*. Rather than sending messages directly, the principal layer in (busy) primary nodes uses the service of the channel layer for communication. This service, called *channels*, bears similarities to virtual circuits such as TCP connections. As in the case of TCP connections, a channel χ between two primary nodes $p, p' \in P$ can be released, and sometimes later, a new channel χ' between p and p' may be created. An important feature of the channels is that a message sent from p to p' as part of χ does not arrive as part of χ'; this is formalized, together with several other important assurances of the channel layer, in Grnt. 1–6 below.

A reader who is familiar with the nuts and bolts of virtual circuits knows that the task of ensuring such properties is messy, sometimes non-trivial, but nonetheless possible. This is the reason we defer the description of the implementation of the channel layer to the full version [22], whereas the principal layer's implementation is presented in the current section. For now, let us just say that in the channel layer, each primary node $p \in P$ updates all the utility nodes in $U(p)$ whenever its status changes from busy to non-busy or vice versa. A (non-faulty) utility node $u \in U$ chooses two primary nodes in $P(u)$ that reported they are busy, and creates a channel between them. This channel is released by u when (and only when) u hears that one of the channel's primary nodes is no longer busy. We now provide a more formal description of the channels and the interface between the two layers.

The Channels. The role of the channels is to enable (duplex) communication among (unordered) pairs of busy primary nodes. For $p, p' \in P$, each $\{p, p'\}$-channel χ is associated with a utility node $u \in U(p) \cap U(p')$, referred to as the channel's *mediator* that relays the messages that p and p' exchange with each other as part of χ, referred to hereafter as *relayed messages*. The algorithm is designed so that for every $u \in U$ and $p, p' \in P$, at any given time, the system includes at most one channel mediated by u and zero or more $\{p, p'\}$-channels, each mediated by its own utility node in $U(p) \cap U(p')$.

To keep track of the channels, a primary node $p \in P$ maintains the local variable $p.\texttt{meds} \subseteq U(p)$ that stores the mediators of the $\{p, \cdot\}$-channels; a utility node $u \in U$ maintains the local variable $u.\texttt{chan} \subseteq P(u)$ defined so that $u.\texttt{chan} = \{p, p'\}$ if u mediates a $\{p, p'\}$-channel, and $u.\texttt{chan} = \emptyset$ otherwise. For a $\{p, p'\}$-channel χ mediated by u, we refer to the time $t > 0$ at which u sets $u.\texttt{chan} \leftarrow \{p, p'\}$ as the *creation time* of χ; we refer to the earliest time $\bar{t} > t$ such that $u.\texttt{chan}^{\bar{t}+} \neq \{p, p'\}$ as the *release time* of χ.

We emphasize that channel χ is created (resp., released) once and if the mediator u creates (resp., releases) a $\{p, p'\}$-channel χ' at time $t' \neq t$ (resp., $\bar{t}' \neq \bar{t}$), then χ' and χ are considered to be two different channels. Notice that the placeholders χ and χ' are introduced for the sake of the discussion and we do not assume that the primary nodes p and p' agree on "common names" for the $\{p, p'\}$-channels.

[13] Throughout, we omit the superscript t from $\texttt{obj}^t$ when we wish to address the dynamic nature of the object $\texttt{obj}$ whose state may vary over time.

Consider some primary node $p \in P$ and utility node $u \in U(p)$. A key feature of the algorithm is that if $I \subset \mathbb{R}_{>0}$ is a maximal time interval such that $u \in p.\texttt{meds}^t$ for all $t \in I$, then all relayed messages that p sends to or receives from u during I belong to the same $\{p, p'\}$-channel χ (mediated by u) for some primary node $p' \in P(u) - \{p\}$; we refer to χ and p' as the *u-channel* and *u-peer*, respectively, of p during I. Notation-wise, let $\text{channel}^t(p, u)$ and $\text{peer}^t(p, u)$ be the operators that return the u-channel and u-peer, respectively, of p at time t if $u \in p.\texttt{meds}^t$, and $\perp$ otherwise.

The algorithm is designed so that if χ is a $\{p, p'\}$-channel mediated by u, then the set $\{t \in \mathbb{R}_{>0} : \text{channel}^t(p, u) = \chi\}$ is either empty or forms a single interval of the time axis. This is in contrast to the set $\{t \in \mathbb{R}_{>0} : \text{peer}^t(p, u) = p'\}$ that may form multiple intervals, each corresponding to a different $\{p, p'\}$-channel mediated by u.

Notice that $\text{channel}^t(p, u) = \chi \neq \perp$ does not imply that channel χ still exists at time t as χ may have been released by u at time $t - 1 \leq t' < t$ (which will be observed by p at time $t' < t'' \leq t' + 1$). Moreover, while one may hope that the formula $\text{peer}^t(p, u) = p' \iff \text{peer}^t(p', u) = p$ is satisfied "most of the time", it cannot be satisfied all the time; indeed, given a $\{p, p'\}$-channel χ, since the variables $p.\texttt{meds}$ and $p'.\texttt{meds}$ are updated asynchronously, we cannot expect the aforementioned formula to be satisfied "shortly after" (resp., "shortly before") χ is created (resp., released). Grnt. 4 ensures that these inconsistencies do not introduce "misunderstandings".

The Interface between the Layers. The channel layer is responsible for maintaining the channels by updating the `meds` and `chan` variables and for handling the delivery of relayed messages among peers. The principal layer governs the token gathering process among the busy primary nodes that communicate with their peers over the channels. For a primary node $p \in P$, the main component in the interface between the two layers is the set $\mathcal{C}^t(p) = \{\text{channel}^t(p, u) : u \in p.\texttt{meds}^t\}$ that captures p's channels at time $t > 0$. It is assumed that node p's channel layer maintains the set $\mathcal{C}(p)$ while hiding its implementation details; node p's principal layer then uses the channels in $\mathcal{C}(p)$ to exchange relayed messages with the principal layer of p's peers.

A primary node $p \in P$ is regarded as *operational* at time $t > 0$ if $\mathcal{C}^t(p) \neq \emptyset$; let $\mathcal{OP}^t$ be the set of operational primary nodes at time t. A $\{p, p'\}$-channel χ is regarded as *operational* at time $t > 0$ if $\chi \in \mathcal{C}^t(p) \cap \mathcal{C}^t(p')$; let $\mathcal{OC}^t$ be the set of operational channels at time t. These notions facilitate the formulation of the key assurances that the channel layer provides to the principal layer, stated in Grnt. 1–6 (established in the full version [22]).

▶ **Guarantee 1.** *$\mathcal{OP}^t \subseteq \mathcal{B}^t$ for every $t > 0$.*

▶ **Guarantee 2.** *For every $t_0 > 0$, if $|\{p \in P : p \in \mathcal{B}^t \text{ for all } t_0 < t \leq t_0 + 4\}| \geq 2$, then there exists $t_0 < t \leq t_0 + 4$ such that $\mathcal{OC}^{t+} \neq \emptyset$.*

▶ **Guarantee 3.** *Consider a $\{p, p'\}$-channel χ and time $t > 0$, and assume that $\chi \in \mathcal{OC}^t$. Then, $\chi \in \mathcal{OC}^{t+}$ if and only if $p, p' \in \mathcal{B}^{t+}$. Moreover, if $p \notin \mathcal{B}^{t+}$, then $\chi \notin \mathcal{C}^{(t+2)+}(p')$.*

▶ **Guarantee 4.** *The following conditions are satisfied for every primary node $p \in P$, time $t > 0$, and $\{p, p'\}$-channel $\chi \in \mathcal{C}^t(p)$: (I) If the principal layer of p receives a relayed message μ over χ at time t, then μ was sent over χ by the principal layer of p' at time $t - 2 \leq t' < t$ (in other words, μ could not have originated from some past/future channel of p with the same mediator). (II) If the principal layer of p sends a relayed message μ over χ at time t, then either (II.a) μ is received over χ by the principal layer of p' at time $t < t' \leq t + 2$; or (II.b) μ becomes irrelevant because p' turned from busy to non-busy, leading to the removal of χ from $\mathcal{C}(p')$.*

▶ **Guarantee 5.** $|\mathcal{C}^t(p)| \leq O(\sqrt{n \log n})$ *for every* $p \in P$ *and* $t > 0$.

▶ **Guarantee 6.** *The channel layer sends* $O(\sqrt{n \log n})$ *messages per token.*

The Principal Layer

We now turn to describe the implementation of the principal layer, building on the interface assured by Grnt. 1–6 (pseudocode is provided in the full version [22]). Recall that this layer is implemented over the (busy) primary nodes $p \in P$ that exchange relayed messages with their peers over the channels in $\mathcal{C}(p)$ (maintained by the channel layer); the utility nodes do not participate in this layer and are abstracted away from its description. The policy of the principal layer for a non-operational node $p \in P$ is straightforward: p performs (as many as possible) team formation operations if $p.\texttt{tok} \geq \sigma$, and does nothing otherwise. The interesting part of the principal layer algorithm addresses the operational nodes that "compete" with their peers over the right to serve as the local gathering point for the tokens. To this end, if $p.\texttt{tok}, p'.\texttt{tok} < \sigma$ for some $p, p' \in \mathcal{OP}$ and $\mathcal{C}(p) \cap \mathcal{C}(p') \neq \emptyset$, then (local) symmetry breaking is needed to determine whether p transports its tokens to p' or p' transports its tokens to p (or neither).

The principal layer resolves this symmetry breaking challenge by following a "star-shaped" token transportation pattern. Specifically, an operational node $p \in \mathcal{OP}$ runs consecutive *phases* so that in each phase, p assumes a *phase type* that can be either *center* or *arm*. This phase type is determined by p via an (unbiased) coin toss when the phase begins and is stored in the local variable $p.\texttt{phase-type} \in \{\mathsf{center}, \mathsf{arm}\}$ that p maintains. The crux of the (dynamic) classification into phase types is that a center node attempts to collect the tokens from its arm peers so that a phase of a center node p is successful if one or more of p's arm peers transport their tokens to p; a phase of an arm node p is successful if p transports its tokens to exactly one of its center peers.

Each phase ϕ of node p proceeds according to the following *request-response mechanism*: When ϕ begins, node p sends a *request message* over each channel in $\mathcal{C}(p)$; phase ϕ ends once p has received a *response message* for each request message sent at the beginning of ϕ. (As discussed soon, ϕ may end abruptly if p decides to transport its tokens to one of its peers, in which case, p becomes non-busy and hence, also non-operational.) To implement this request-response mechanism, p maintains the local variable $p.\texttt{awaiting-resp}(\chi) \in \{\mathsf{true}, \mathsf{false}\}$ for each $\chi \in \mathcal{C}(p)$, setting $p.\texttt{awaiting-resp}(\chi) \leftarrow \mathsf{true}$ when a request message is sent over χ and resetting $p.\texttt{awaiting-resp}(\chi) \leftarrow \mathsf{false}$ when a response message is received over χ.

When phase ϕ ends, node p checks if the inequality $p.\texttt{tok} \geq \sigma$ holds and if it does, performs (as many as possible) team formation operations. To simplify the presentation, we treat any tokens injected into p during phase ϕ as if they are injected when ϕ ends; we emphasize that the injected tokens are accounted for when checking if the inequality $p.\texttt{tok} \geq \sigma$ holds.[14]

The type of the request messages depends on the phase type of their sender: the requests of the center nodes are TokensPlease messages, whereas the requests of the arm nodes are Waiting messages. The reaction of the operational nodes to an incoming request, including the type of the corresponding response message, depends on the phase type of the receiver as well as on the type of the incoming request. To this end, consider an operational node $p \in \mathcal{OP}$ that receives a request μ over channel $\chi \in \mathcal{C}(p)$ during phase ϕ.

[14] To adhere to the formal event driven model defined in Sec. 3.1, that forbids concurrent activation events, one can move the token injection event "shortly after" the end of phase ϕ so that this event triggers the team formation operations (if any) or the beginning of the subsequent phase (if p is still operational).

Assume first that $\mu = \mathsf{TokensPlease}$. If ϕ is a center phase, then p reacts by sending a $\mathsf{NoTransport}$ response. If ϕ is an arm phase, then p reacts by sending a $\mathsf{Transport}(p.\mathtt{tok})$ response that transports the tokens held by p (excluding those injected during ϕ) to the peer of p at the other end of χ. Consequently, p becomes non-busy and the channels in $\mathcal{C}(p)$ are released (the channel layer takes care of that), making p non-operational; this is the one exception to the request-response mechanism, where phase ϕ ends abruptly without waiting for all the responses (since $\mathcal{C}(p)$ is emptied, Grnt. 4 ensures that the "missing responses" are dropped and do not interfere with future channels of p).

Now, assume that $\mu = \mathsf{Waiting}$. If ϕ is an arm phase, then p reacts by sending a GoOn response. If ϕ is a center phase, then p delays its response until the current phase ϕ ends and the phase type of the next phase ϕ' of p is revealed. At this stage, if ϕ' is a center phase, then p "forgets about" μ and continues as usual with ϕ'; in particular, p sends $\mathsf{TokensPlease}$ requests over all channels in $\mathcal{C}(p)$, including χ – this $\mathsf{TokensPlease}$ request over χ plays a key role because it is guaranteed to prompt the peer p' of p on the other end of χ to transport its tokens if p' did not do so already. If ϕ' is an arm phase, then p first sends a GoOn message over χ and then, continues as usual with ϕ' (sending $\mathsf{Waiting}$ requests over all channels in $\mathcal{C}(p)$). To implement this policy, p maintains the local variable $p.\mathtt{delaying\text{-}resp}(\chi) \in \{\mathsf{true}, \mathsf{false}\}$ for each channel $\chi \in \mathcal{C}(p)$, setting $p.\mathtt{delaying\text{-}resp}(\chi) \leftarrow \mathsf{true}$ when p receives a $\mathsf{Waiting}$ message over χ as part of a center phase ϕ and resetting $p.\mathtt{delaying\text{-}resp}(\chi) \leftarrow \mathsf{false}$ at the beginning of the next phase ϕ', after a GoOn response was sent over χ if needed (i.e., if ϕ' is an arm phase).

We emphasize that the $p.\mathtt{awaiting\text{-}resp}(\chi)$ and $p.\mathtt{delaying\text{-}resp}(\chi)$ variables of p are maintained only for channels $\chi \in \mathcal{C}(p)$. In particular, if channel χ is added to (resp., removed from) $\mathcal{C}(p)$ in the midst of phase ϕ, then the variables $p.\mathtt{awaiting\text{-}resp}(\chi)$ and $p.\mathtt{delaying\text{-}resp}(\chi)$ are created (resp., deleted) with it, where both of them are initialized to false.

5 Analysis of the Algorithm – Correctness and Message Load

Throughout this section, we fix some adversarial policy $\mathtt{Adv}$ and analyze the execution of algorithm $\mathtt{Alg}_{\mathrm{TF}}$ under $\mathtt{Adv}$. We start with some useful definitions, followed by Obs. 2 and 3, that capture basic features of $\mathtt{Alg}_{\mathrm{TF}}$, and Lem. 4, that serves as the cornerstone of the entire analysis. Sec. 5.1 is then dedicated to establishing the algorithm's correctness while the message load analysis is presented in Sec. 5.2. The reaction time analysis is deferred to as it builds on a certain implementation feature of the channel layer, presented in Throughout the analysis, we condition on the event that the assertion of Lem. 1 holds. Proofs are omitted from this extended abstract and appear in the full version [22].

We say that a primary node $p \in P$ *retires* at time $t > 0$ if $p \in \mathcal{OP}^t - \mathcal{B}^{t+}$, observing that this holds if and only if a phase ϕ of p ends at time t and either (1) ϕ is an arm phase that ends abruptly and p sends a $\mathsf{Transport}$ message at time t; or (2) p performs team formation(s) at time t. We say that a $\{p_1, p_2\}$-channel χ *retires* at time $t > 0$ if $\chi \in \mathcal{OC}^t - \mathcal{OC}^{t+}$, observing that by Grnt. 3, this holds if and only if $\chi \in \mathcal{OC}^t$ and p_i retires at time t for some (exactly one) $i \in \{1, 2\}$. Notice that if a channel χ becomes operational (resp., retires) at time $t > 0$ (resp., at time $t' > t$), then χ is created (resp., released) by its mediator strictly before time t (resp., strictly after time t').

For $p \in P$ and $t > 0$, let $\Phi_p(t)$ be the operator that returns the earliest time $t' > t$ such that a phase of p ends at time t' if $p \in \mathcal{OP}^{t+}$, and t otherwise (Obs. 2 ensures that this operator is well defined). Notice that if we fix time $t > 0$, including all coin tosses up to

(excluding) time t, then $\Phi_p(t)$ is a random variable that depends on the coin tosses from time t onward. Notice further that while the condition $p \in \mathcal{OP}^t \wedge \Phi_p(t) = t$ does not imply that p retires at time t (it may be the case that p is no longer operational at time $t+$ although it is still busy), this condition does imply that every channel $\chi \in \mathcal{C}^t(p)$ retires at or before time t.

The $\Phi_p(\cdot)$ operator is extended inductively as follows: let $\Phi_p^0(t) = t$; for $i > 0$, let $\Phi_p^i(t) = \Phi_p(\Phi_p^{i-1}(t))$. This extension gives us a convenient handle for reasoning about events that occur "within the next i phases" of p.

▶ **Observation 2.** *A phase that begins at time $t > 0$ is guaranteed to end at or before time $t + 8$.*

▶ **Observation 3.** *If a* Transport *response is in transit from a primary node p to a primary node p' over a $\{p, p'\}$-channel χ at time $t > 0$, then $p' \in \mathcal{B}^t$ and $\chi \in \mathcal{C}^t(p')$.*

▶ **Lemma 4.** *Fix time $t_0 > 0$ and consider a $\{p_1, p_2\}$-channel $\chi \in \mathcal{OC}^{t_0}$. With probability at least $1/16$, channel χ retires during the time interval $[t_0, \min_{i\in\{1,2\}} \Phi_{p_i}^3(t_0)]$.*

5.1 Safety and Liveness

In this section, we establish the correctness of $\texttt{Alg}_{\text{TF}}$, proving that it satisfies the safety and liveness conditions. The former condition holds trivially as tokens are deleted only during team formation operations, so the remainder of this section is dedicated to the latter.

Consider time $t_0 > 0$ and assume that the system contains at least σ tokens at time t_0. For time $t \geq t_0$, node $p \in P$, and $\{p, p'\}$-channel $\chi \in \mathcal{C}^t(p)$, let $R^t(p, \chi)$ be the number of tokens transported over χ towards p at time t. Notice that $R^t(p, \chi) = k > 0$ if and only if there is a Transport(k) message in transit from p' to p over χ at time t. For time $t \geq t_0$ and node $p \in P$, let $R^t(p) = \sum_{\chi\in\mathcal{C}^t(p)} R^t(p, \chi)$ if $p \in \mathcal{B}^t$, and $R^t(p) = 0$ otherwise.

By Obs. 2, a primary node that holds at least σ tokens is guaranteed to form a team in $O(1)$ time. Grnt. 4 ensures that every Transport message reaches its destination in $O(1)$ time. Therefore, if $p.\texttt{tok}^t + R^t(p) \geq \sigma$, then p is guaranteed to form a team by time $t + O(1)$.

Define the potential function $\psi : \mathbb{R}_{>0} \to \mathbb{Z}_{\geq 0}$ as

$$\psi(t) = \sum_{p\in P} (p.\texttt{tok}^t + R^t(p) - 1)$$

and notice that if $\psi(t) \geq (\sigma - 1)n$, then $p.\texttt{tok}^t + R^t(p) \geq \sigma$ for some $p \in P$, hence a team is guaranteed to be formed by time $t + O(1)$. Obs. 3 ensures that if no teams are formed during the time interval $I = [t_0, \cdot] \subset \mathbb{R}_{>0}$, then the function $\psi(t)$ is non-decreasing in I. Moreover, a primary node $p \in P$ retires at time $t \geq t_0$ without forming any team if and only if p sends a Transport message at time t, implying that $\psi(t+) = \psi(t) + 1$. So, it suffices to show that as long as no teams are formed, the number of node retirements must increase.

By Obs. 2 and Lem. 4, we know that if $\mathcal{OC}^t \neq \emptyset$, then at least one primary node is certain to retire in finite time with probability 1. The liveness proof is completed by observing that if the system contains $k \geq \sigma$ tokens and there are no tokens in transit, then either (I) all k tokens are held by a single busy node that forms a team in $O(1)$ time; or (II) the k tokens are distributed over multiple busy nodes, in which case, an operational channel is certain to be generated in $O(1)$ time by Grnt. 2.

▶ **Theorem 5.** $\texttt{Alg}_{\text{TF}}$ *satisfies the safety and liveness conditions for any TF instance.*

5.2 Message Load

We now turn to analyze the message load of $\texttt{Alg}_{\text{TF}}$ in expectation and whp. To this end, fix some $\ell \in \mathbb{Z}_{>0}$ and assume that the adversarial policy $\texttt{Adv}$ injects at most ℓ tokens throughout the execution (i.e., $\texttt{Adv} \in \mathcal{A}_\ell$ in the language of Sec. 3.1). The analysis starts with bounding the number of relayed messages sent over non-operational channels.

▶ **Lemma 6.** *The total number of relayed messages sent throughout the execution over channels χ while $\chi \notin \mathcal{OC}$ is $O(\ell \cdot \sqrt{n \log n})$.*

Combined with Grnt. 6, we conclude that with the exception of relayed messages sent over operational channels, the total number of messages sent by $\texttt{Alg}_{\text{TF}}$ is $O(\ell \cdot \sqrt{n \log n})$, so it remains to bound the number of former messages. To this end, we establish Lem. 7 and Lem. 8.

▶ **Lemma 7.** $\left|\bigcup_{t>0} \mathcal{OC}^t\right| \leq O(\ell \cdot \sqrt{n \log n})$.

▶ **Lemma 8.** *Consider a $\{p, p'\}$-channel χ and let Y_χ be a random variable that counts the number of relayed messages sent over χ while $\chi \in \mathcal{OC}$. There exist constants $\alpha, \beta > 0$ such that $\mathbb{P}(Y_\chi > y) < \beta e^{-\alpha y}$ for every $y \in \mathbb{Z}_{\geq 0}$.*

For a channel χ, let Y_χ be the random variable in Lem. 8. Recalling that the expected value of an exponentially decaying random variable is bounded by $O(1)$, we can combine Lem. 7 and 8 to deduce, by the linearity of expectation, that $\texttt{Alg}_{\text{TF}}$ sends $O(\ell \cdot \sqrt{n \log n})$ messages in expectation.

At this stage, one may be tempted to advance towards a whp bound by assuming that the random variables Y_χ of different channels χ are independent. Unfortunately, this assumption is wrong: the number of relayed messages sent over different channels in $\mathcal{OC}$ may be strongly correlated. To resolve this difficulty, we use the following lemma, stating that the average of finitely many exponentially decaying random variables is also exponentially decaying even if the original random variables exhibit complex dependencies; we will not be surprised to learn that this lemma is known from the existing literature, however, we could not find it anywhere and therefore, provide a standalone proof (deferred to the full version [22]).

▶ **Lemma 9.** *Fix some $\alpha, \beta > 0$ and let $X_1, \ldots X_m$ be (not necessarily independent) random variables over $\mathbb{R}_{>0}$ such that $\mathbb{P}(X_i > x) < \beta e^{-\alpha x}$ for every $i \in [m]$ and $x > 0$. For every $\epsilon > 0$, there exists $\beta' = \beta'(\beta, \epsilon) > 0$ such that $\mathbb{P}(\bar{X} > x) < \beta' e^{-\frac{\alpha x}{1+\epsilon}}$ for every $x > 0$, where $\bar{X} = \frac{1}{m} \sum_{i \in [m]} X_i$.*

Let $m = O(\ell \cdot \sqrt{n \log n})$ be the bound promised in Lem. 7. By applying Lem. 9 to the (at most) m random variables in $\{Y_\chi\}_{\chi \in \bigcup_{t>0} \mathcal{OC}^t}$, we conclude that $\frac{1}{m} \sum_{\chi \in \bigcup_{t>0} \mathcal{OC}^t} Y_\chi \leq O(\log n)$ whp, hence $\texttt{Alg}_{\text{TF}}$ sends $O(\ell \cdot \sqrt{n \log n} \cdot \log n)$ messages whp.

▶ **Theorem 10.** *The message load of $\texttt{Alg}_{\text{TF}}$ is $O(\sqrt{n \log n})$ in expectation and $O(\sqrt{n \log n} \cdot \log n)$ whp.*

References

1 Yehuda Afek, Baruch Awerbuch, Serge A. Plotkin, and Michael E. Saks. Local management of a global resource in a communication network. *J. ACM*, 43(1):1–19, 1996. doi:10.1145/227595.227596.

2 Yehuda Afek and Eli Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. *SIAM J. Comput.*, 20(2):376–394, 1991. doi:10.1137/0220023.

3 Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Inf. Comput.*, 113(2):312–330, 1994. doi:10.1006/INCO.1994.1075.

4 Yehuda Afek and Michael E. Saks. Detecting global termination conditions in the face of uncertainty. In Fred B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 109–124. ACM, 1987. doi:10.1145/41840.41850.

5 Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006. doi:10.1137/050645221.

6 Dana Angluin. Local and global properties in networks of processors (extended abstract). In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.

7 Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004.

8 John Augustine, Anisur Rahaman Molla, and Gopal Pandurangan. Sublinear message bounds for randomized agreement. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 315–324. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212751.

9 Baruch Awerbuch, Lenore Cowen, and Mark A. Smith. Efficient asynchronous distributed symmetry breaking. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 214–223. ACM, 1994. doi:10.1145/195058.195136.

10 Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *Journal of the ACM (JACM)*, 37(2):238–256, 1990. doi:10.1145/77600.77618.

11 Reuven Bar-Yehuda, Shay Kutten, Yaron Wolfstahl, and Shmuel Zaks. Making distributed spanning tree algorithms fault-resilient. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS 87, 4th Annual Symposium on Theoretical Aspects of Computer Science, Passau, Germany, February 19-21, 1987, Proceedings*, volume 247 of *Lecture Notes in Computer Science*, pages 432–444. Springer, 1987. doi:10.1007/BFB0039625.

12 Bonnie Bassler. *Quorum Sensing: How Bacteria Communicate*. The Explorer's Guide to Biology, September 2019. 48 pages; https://explorebiology.org/summary/cell-biology/quorum-sensing:-how-bacteria-communicate.

13 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

14 Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984. doi:10.1145/800222.806743.

15 Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 12–26. ACM, 1983. doi:10.1145/800221.806706.

16 Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, Vijay K. Garg, and Yogish Sabharwal. Efficient decentralized algorithms for the distributed trigger counting problem. *Theory Comput. Syst.*, 51(4):447–473, 2012. doi:10.1007/S00224-012-9405-4.

17 Che-Cheng Chang and Jichiang Tsai. Distributed trigger counting algorithms for arbitrary network topology. *Wirel. Commun. Mob. Comput.*, 16(16):2463–2476, 2016. doi:10.1002/WCM.2698.

18 Edsger W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965. doi:10.1145/365559.365617.

19 Stefan Dobrev, Rastislav Kralovic, Nicola Santoro, and Wei Shi. Black hole search in asynchronous rings using tokens. In Tiziana Calamoneri, Irene Finocchi, and Giuseppe F. Italiano, editors, *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, volume 3998 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2006. doi:10.1007/11758471_16.

20 Yuval Emek and Amos Korman. Efficient threshold detection in a distributed environment: extended abstract. In Andréa W. Richa and Rachid Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 183–191. ACM, 2010. doi:10.1145/1835698.1835742.

21 Yuval Emek and Amos Korman. New bounds for the controller problem. *Distributed Comput.*, 24(3-4):177–186, 2011. doi:10.1007/S00446-010-0119-Z.

22 Yuval Emek, Shay Kutten, Ido Rafael, and Gadi Taubenfeld. Team formation and applications, 2025. doi:10.48550/arXiv.2508.13084.

23 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344. ACM, 2016. doi:10.1145/2897518.2897557.

24 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

25 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. doi:10.1016/J.TCS.2005.01.001.

26 Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/NET.20127.

27 W. Claiborne Fuqua, Stephen C. Winans, and E. Peter Greenberg. Quorum sensing in bacteria: the LuxR-LuxI family of cell density-responsive transcriptional regulators. *Journal of Bacteriology*, 176(2):269–275, 1994.

28 Rahul Garg, Vijay K. Garg, and Yogish Sabharwal. Scalable algorithms for global snapshots in distributed systems. In Gregory K. Egan and Yoichi Muraoka, editors, *Proceedings of the 20th Annual International Conference on Supercomputing, ICS 2006, Cairns, Queensland, Australia, June 28 - July 01, 2006*, pages 269–277. ACM, 2006. doi:10.1145/1183401.1183439.

29 Seth Gilbert and Dariusz R. Kowalski. Distributed agreement with optimal communication complexity. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 965–977. SIAM, 2010. doi:10.1137/1.9781611973075.78.

30 Sergiu Hart. Games in extensive and strategic forms. In *Handbook of Game Theory with Economic Applications*, volume 1, pages 19–40. Elsevier, 1992. doi:10.1016/S1574-0005(05)80005-0.

31 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Comput.*, 28(1):31–53, 2015. doi:10.1007/S00446-013-0202-3.

32 Ling Huang, Minos N. Garofalakis, Anthony D. Joseph, and Nina Taft. Communication-efficient tracking of distributed cumulative triggers. In *27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007), June 25-29, 2007, Toronto, Ontario, Canada*, page 54. IEEE Computer Society, 2007. doi:10.1109/ICDCS.2007.93.

33 Pierre A. Humblet. *Selecting a leader in a clique in 0 (N log N) messages*. Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1984.

34 Alon Itai, Shay Kutten, Yaron Wolfstahl, and Shmuel Zaks. Optimal distributed t-resilient election in complete networks. *IEEE Trans. Software Eng.*, 16(4):415–420, 1990. doi:10.1109/32.54293.

35 Prasad Jayanti, Siddhartha Jayanti, and Sucharita Jayanti. Towards an ideal queue lock. In Nandini Mukherjee and Sriram V. Pemmaraju, editors, *ICDCN 2020: 21st International Conference on Distributed Computing and Networking, Kolkata, India, January 4-7, 2020*, pages 19:1–19:10. ACM, 2020. doi:10.1145/3369740.3369784.

36 Ram Keralapura, Graham Cormode, and Jeyashankher Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 289–300. ACM, 2006. doi:10.1145/1142473.1142507.

37 Seokhyun Kim, Giorgia Fattori, and Yongsu Park. A simple and efficient distributed trigger counting algorithm based on local thresholds. *ICT Express*, 10(4):895–901, 2024. doi:10.1016/J.ICTE.2024.05.005.

38 Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 199–207. ACM, 1984. doi:10.1145/800222.806747.

39 Amos Korman and Shay Kutten. Controller and estimator for dynamic networks. In Indranil Gupta and Roger Wattenhofer, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 175–184. ACM, 2007. doi:10.1145/1281100.1281127.

40 Manish Kumar and Anisur Rahaman Molla. On the message complexity of fault-tolerant computation: Leader election and agreement. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 294–295. ACM, 2023. doi:10.1145/3571306.3571424.

41 Manish Kumar and Anisur Rahaman Molla. Sublinear message bounds of authenticated implicit byzantine agreement. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN 2024, Chennai, India, January 4-7, 2024*, pages 124–133. ACM, 2024. doi:10.1145/3631461.3631548.

42 Shay Kutten. Optimal fault-tolerant distributed construction of a spanning forest. *Inf. Process. Lett.*, 27(6):299–307, 1988. doi:10.1016/0020-0190(88)90217-7.

43 Shay Kutten, William K. Moses Jr., Gopal Pandurangan, and David Peleg. Singularly near optimal leader election in asynchronous networks. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 27:1–27:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.27.

44 Shay Kutten, William K Moses Jr, Gopal Pandurangan, and David Peleg. Singularly optimal randomized leader election. In *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179, 2020. doi:10.4230/LIPICS.DISC.2020.22.

45 Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theor. Comput. Sci.*, 561:134–143, 2015. doi:10.1016/J.TCS.2014.02.009.

46 Shay Kutten, Peter Robinson, Ming Ming Tan, and Xianbin Zhu. Improved tradeoffs for leader election. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 355–365. ACM, 2023. doi:10.1145/3583668.3594576.

47 Leslie Lamport. A new solution of dijkstra's concurrent programming problem. *Commun. ACM*, 17(8):453–455, 1974. doi:10.1145/361082.361093.

48 Giuseppe Antonio Di Luna, Ryuhei Uehara, Giovanni Viglietta, and Yukiko Yamauchi. Gathering on a circle with limited visibility by anonymous oblivious robots. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.12.

49 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

50 Earl H. McKinney. Generalized birthday problem. *The American Mathematical Monthly*, 73(4):385–387, 1966.

51 El Mahdi El Mhamdi, Rachid Guerraoui, Alexandre Maurer, and Vladislav Tempez. Exploring the borderlands of the gathering problem. *Bull. EATCS*, 129, 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/593/602.

52 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

53 Stephen C. Pratt. Quorum sensing by encounter rates in the ant Temnothorax albipennis. *Behavioral Ecology*, 16(2):488–496, 2005.

54 Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.48.

55 Red Hat Publication. Stateful vs. stateless applications. Accessed online, February 5th, 2025. URL: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless#:~:text=Stateless%20applications%20are%20generally%20more,it%20more%20difficult%20to%20scale.

56 Thomas D. Seeley and P. Kirk Visscher. Group decision making in nest-site selection by honey bees. *Apidologie*, 35(2):101–16, 2004.

57 Eugene Styer and Gary L. Peterson. Tight bounds for shared memory symmetric mutual exclusion problems. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 177–191. ACM, 1989. doi:10.1145/72981.72993.

58 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

59 Gadi Taubenfeld. *Synchronization Algorithms and Concurrent Programming*. Pearson / Prentice-Hall, 2006. ISBN 0-131-97259-6, 423 pages.

60 Gadi Taubenfeld, Shmuel Katz, and Shlomo Moran. Initial failures in distributed computations. *Int. J. Parallel Program.*, 18(4):255–276, 1989. doi:10.1007/BF01407859.

61 Masafumi Yamashita and Tiko Kameda. Computing on an anonymous network. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 117–130. ACM, 1988. doi:10.1145/62546.62568.

Appendix

A Applications – Additional Information

In this section, we provide further technical details of some of the applications discussed in Sec. 1.4. The details of the other applications are omitted from this extended abstract and can be found in the full version [22].

A.1 Leader Election

A.1.1 Implicit leader election

Implicit LE is defined in Sec. 1.4. We first derive a simpler algorithm $LE-I1$ that has no termination detection for some nodes and add the termination detection later.

Informally, it is assumed that there are $(1/2+\epsilon)n$ non-fragile nodes and each of them wakes up and starts the algorithm. Any fragile node may also be set by the adversary to be non-faulty at that point and start the algorithm, too.

1. Each node starting the algorithm flips a coin with probability $\frac{c\log n}{n}$ (for some constant c that is "large enough"; the selection of c uses rather standard considerations, so we skip it here). Let $Cand$ be the set of nodes that flipped 1.
2. Each node not in $Cand$ sets its status to "not-leader."
3. Each node in $Cand$ generates a token and starts the algorithm for TF with $\sigma = (1-\frac{\epsilon}{2})(\frac{1}{2}+\epsilon)c\log n$ (rounded up, let us ignore that for simplicity of notation).
4. A node that deleted σ tokens in TF sets its status to "leader."

The following observation follows directly from Thm. 1 and the algorithm. (For the second part, note that $\sigma = O(\log n)$.)

▶ **Observation 11.** *Algorithm $LE-I1$ satisfies the following two properties:*

- *A node that is not the elected leader may not know that it is not the leader. (We shall fix this later.)*
- *The message complexity of Algorithm $LE-I1$ is $O(\sqrt{n\log n})$ in expectation and $O(\sqrt{n\log n}\cdot\log n)$ whp. The time complexity is $O(\log n)$.*

▶ **Lemma 12.** *Algorithm $LE-I1$ elects a unique leader whp.*

Proof. Let us analyze the relation between σ and the number T of tokens generated by nodes in the third line of the algorithm. The number n' of nodes that flipped coins may include any number (between zero and $(1/2-\epsilon)n$) of fragile nodes; hence, $(1/2+\epsilon)n \le n' \le n$. Thus, for the expected number $\frac{cn'\log n}{n}$ of tokens generated, we know that $(1/2+\epsilon)c\log n \le \frac{cn'\log n}{n} \le c\log n$. Applying Chernoff's bound, we find for the random variable T (the number of tokens) that whp,

$$(1-\frac{\epsilon}{2})(\frac{1}{2}+\epsilon)c\log n \le T \le (1+\frac{\epsilon}{2})c\log n.$$

By the left inequality, $\sigma \le T$ whp. Hence, at least one set of tokens can be deleted in TF, which means that at least one node is elected in step 4 of the algorithm.

It remains to show that $T < 2\sigma$, so no more than one deletion event occurs (whp) and thus, no more than one leader is elected. By the above bounds on the number of tokens, $T/\sigma = \frac{T}{(1-\epsilon/2)(1/2+\epsilon)c\log n} \le \frac{(1+\epsilon/2)c\log n}{(1-\epsilon/2)(1/2+\epsilon)c\log n} \le \frac{(1+\epsilon/2)}{(1-\epsilon/2)(1/2+\epsilon)}$. This is strictly smaller than 2. ◀

Termination Detection for Non-Leaders. Let us now show how to enhance algorithm $LE-I1$ such that every node that is *not* elected will set its status to "not-leader" eventually, as is sometimes required. That is, the promised $LE-I$ algorithm will first run Algorithm $LE-I1$ (that runs the TF algorithm). After that, the elected leader will invoke a termination detection module $Term$.

We came up with several methods of implementing $Term$. Let us describe two different implementations. The first one uses the accumulation property of the algorithm (see

Sec. 1.3). The second one does not use the accumulation property and is given here to show that a reduction from LE to TF is possible without using additional features. For both implementations, mark every token created in the third step of $LE - I1$ by "LE token". The implementations use an additional kind of token, called TERM (described below). The TF algorithm (run as a subroutine of the LE algorithm) does not distinguish between the two kinds of tokens. However, the $Term$ part of the LE notices when a TERM token participates in a formation (second implementation) or is received (first implementation) and makes use of this knowledge.

First Implementation. The leader r (the node who performed the team formation) creates one TERM token. The underlying TF algorithm treats the TERM token exactly like any other token. However, any non-leader node $v \neq r$ performing $LE - I1$ that receives a TERM token, note that it (v) is not a leader.

Recall that by the accumulation feature of the algorithm, all the tokens that remain in the networks after the team formation eventually are held (forever) by a single node v. If $v \neq r$, then v knows it is not the leader. Let w be any node other than v and r. Either w did not join $Cand$, or it did join but at some point sent its token to another node. In both cases, w knows that it is not the leader. It is easy to show that this implementation does not increase the order of the complexities of the LE election algorithm beyond that of $LE - I1$.

Second Implementation. In this implementation of $Term$, when the leader node deletes σ tokens, it generates $\sigma - 1$ TERM tokens and continues to perform TF. Any other node that deletes σ tokens that include some x TERM tokens and some $\sigma - x$ LE ones generates x TERM ones. Any node that is *not* the leader but ever sees a TERM token, changes its status to "not-leader." The arguments for correctness are similar to those for the first implementation. Note that introducing those new tokens increases the complexities of the algorithm by a logarithmic factor.

A.1.2 Explicit LE

So far, a non-leader node may not know which of its edges leads to the leader. To make the algorithm solve explicit leader election, the elected leader r sends each of its neighbors one message, notifying the neighbor that r is the leader. This adds n messages and a single time unit to the complexities of the election.

B Additional Related Work

The problem of reaching consensus in a complete graph with initial faults was introduced in the seminal paper of [24] since they showed that reaching consensus was impossible in complete asynchronous networks when faults could occur at any time. The same algorithm could be used for leader election. Its message complexity in the CONGEST model was $O(n^3)$ if $O(n)$ nodes could fail. This was improved to $O(n^2)$ in [11] that also addressed general graphs and [34] that solved leader election (and thus also consensus) in a complete graph where the algorithm is given $k < n/2$ that is the maximum possible number of initial faults. The message complexity was $O(n \log n + nk))$ which is $O(n^2)$ in complete networks. For general graphs, a further improvement was obtained in [4] by combining their counting algorithm with an algorithm for election (with no termination detection) in the presence of initial faults, such as the algorithm of [42]. Their message complexity was $O(|E| + n \log^5 n)$. The above mentioned counting was improved in [1], implying a message complexity of

$O(|E| + n \log^3 n)$ for deterministic leader election and consensus under initial faults in general graphs. This means $O(n^2)$ in complete graphs.

Randomization was used to enable consensus in a complete graph in case nodes could fail at any time. Various algorithms were introduced, with various promises on resilience (e.g., also addressing Byzantine faults, varying the number of faults), time complexity, probabilistic assumptions, etc. [13, 15, 14, 54]. They send $\Omega(n^2)$ messages.

Election and consensus in *non-faulty* complete graphs have been studied extensively; see, e.g., [38, 2, 33, 3, 43, 46]. The message complexity in most of them is $\Omega(n \log n)$, with some being $O(n)$, e.g., in the randomized synchronous case. Surprisingly, $o(n)$ (specifically, $O(\log n\sqrt{n})$ was eventually shown for synchronous fault-free networks for the implicit version of the problem (that is, not every node needs to know the output, but no two outputs contradict) [45]. This has been generalized to algorithms that can withstand crash faults [40] or even byzantine faults (assuming authentication) [41] and sublinear message complexity of agreement was also studied [8] but all these was still done in *synchronous* networks. The birthday paradox [50] is used in the above papers; it is used here too, for building a more general structure that supports more general results (such as asynchrony and initial failures).

The *explicit* synchronous case of consensus with crash faults was addressed in [29], using $\Theta(n)$ messages and $\Theta(\log n)$ time.

For the *asynchronous* fault-free case, too, $\Omega(n)$ was considered to be optimal (see, e.g. [43]). It may indeed be for the explicit case, but recall that one of the results in the current paper is an $o(n)$ message result for the *asynchronous* case, even for networks with (generalized) initial faults. The time complexity in [44] for the *non-faulty* synchronous case (and in [43] for the *non-faulty* asynchronous case) is $O(\log^2 n)$, improved here to $O(\log n)$ with at least as good as those of [43, 44] (the message complexity as good as that of the previous synchronous solution and potentially better than the previous asynchronous one).[15]

Various papers addressed problems that are variations of the application we call here distributed trigger counting (DTC). (Some such variations are called threshold detection, threshold sensing, and controller.) The one-shot case is the one where the number of triggers (called "events" in some other papers) passes some threshold [1, 39, 28, 16, 32, 21]. The message complexity obtained was not always analyzed (in the more practical papers), but when analyzed, it was at least $O(\log n \log \sigma)$ even in the case of complete graphs [28]. In the latter case a message could be sent directly from a site (node) to a leader (a coordinator node) that is known in advance. (That model allows sending messages between sites, but the algorithms do not use that.)

The ongoing case is treated, e.g., in [36]. The message complexity per counted event is about the same as in the one-shot case, and some errors in the counting are allowed.

The notion of forgetful algorithms extends the notion of *memoryless* algorithms first defined for deterministic shared-memory mutual exclusion algorithms in [57]. A memoryless mutual exclusion algorithm is an algorithm in which, when all the processes are in their remainder sections, the registers' values are the same as their initial values. This means that a process that tried to enter its critical section did not use any information about its previous attempts (such as the fact that it has entered its critical section five times so far). Almost all known mutual exclusion algorithms are memoryless [59], including Lamport's famous Bakery algorithm [47].

[15] The previous result for the asynchronous case was for general graphs.

C Discussion

We introduce and study the TF problem and demonstrate its usefulness. We consider an asynchronous message-passing system assuming a complete network topology. It would be interesting to study TF in other models of computation assuming, for example, a more general network topology, synchronous systems, a situation where the nodes and/or the tokens are not anonymous, assume that the nodes know the identities of their neighbors (i.e., the KT_1 model [10]), changing the assumption about the message size, etc.

Even for complete networks, interesting problems remain. Can the lower bound be improved? The TF solution helped us improve the known results for some applications. Does it mean that the complexities of these applications are inherently similar to that of TF? or are there upper bounds for some of the applications better than the lower bound for TF? We addressed the case of a small team size σ that seems more practical and challenging; how about the case of a large σ? The complexity per token in the case of a large σ may be much lower. For example, if $\sigma = n + 1$, no team formation is possible unless some node gets at least 2 tokens injected. Hence, it makes sense that an algorithm at a node will do nothing as long as it holds a smaller number of tokens (until approached by another node).

We generalized the TF problem to address a team represented by vectors such as x_1 tokens of color red *and* x_2 tokens of color blue (e.g., x_1 sorcerers and x_2 warriors). How about using other logical operations when composing teams? For example, x_1 sorcerers *and* either x_2 warriors *or* x_3 thieves?

Our algorithms have several novel properties. These properties may be useful for studying other problems, not just TF. One of the properties is, in the context of long-lived problems, resilience to failures *stronger* than initial failures. This might be useful for other long-lived problems. It would be interesting to compare the power of the model that allows such failures to the power of other failure models.

Another important notion we introduced in the context of randomized algorithms is that of a *forgetful* algorithm. While, for deterministic algorithms, this notion is the same as that of a memoryless algorithm, for randomized algorithms, this notion is not as strong as one may expect, as we assume that a node will never forget its initial coin tosses. We use this slightly limited version to ensure our algorithms succeed with high probability, even for infinite executions. Had we sufficed with high probability for each team formation separately, then we would have been able to remove this limitation and ensure that a node does not remember anything (not even its initial coin tosses) in the TF algorithm.[16]

Requiring that nodes forget their past behavior in some cases usually leads to simple solutions, saves the cost of maintaining data structures during quiescent times, and eases the recovery in case of catastrophes, since no data structure needs to be recovered. We emphasize again that in long-lived problems (like mutual exclusion and team formation), a naive solution where a leader is elected and forever coordinates all activities is not forgetful (or memoryless), even in fault-free models.

Let us note that various related notions appeared in the literature. For example, notions of "not keeping information about past behavior" are promoted in practice, e.g., [55], and theory, e.g., [35], because they are considered more scalable and easier to recover.

An interesting property of our TF algorithm seems to be related to privacy and requires further discussion (we have not analyzed this property formally). Even in anonymous networks, nodes in many algorithms (but not the algorithm of this paper) choose identities

[16] When allowing a node to fully forget everything, in an infinite number of team formation where each succeeds whp separately, some formations might not succeed – our slightly limited notion of forgetfulness prevents this situation from happening.

that are unique whp. Suppose that node u sends message M_1 to node v and messages M_2 to node w. Nodes v and w may now compare M_1 and M_2 knowing that they both originated from u. This may harm u's privacy. For example, u may not want other nodes to know that its token participated in a team since this team has some private purpose. Let us say that the algorithm that has this property is *inconspicuous*. It would be interesting to define it formally, prove formally that our algorithm satisfies this property, and study the power of models that allow only such algorithms.

Our generalization of initial failures is not weaker than crash failures. When there are only crash failures, in an infinite execution, a node either never crashes or it is alive only for a finite time (the finite prefix before it crashes). This claim is not valid with our "fragile failures" since there is no limit on how many times the status of a node can be toggled. Another observation regarding crash failures is that TF is not solvable, even with randomization, in the presence of a single crash failure. This observation follows from the fact that a node may crash while holding a token or just before a token is sent to it. (This impossibility holds even with the restriction that nodes with tokens may not crash.) It is possible to modify the liveness condition in the definition of TF so that the above argument does not apply.

Finally, we point out an extensively studied biological phenomenon similar to the one captured by TF. *Quorum sensing* is a process in which bacteria can sense that the number of bacteria ready to release toxins (which cause disease in plants, animals, and humans) has reached a certain threshold. This enables them to release the toxins at approximately the same time [12, 27]. Several insects, like ants and honey bees, have been shown to also use quorum sensing in a process that resembles collective decision-making [53, 56]. Unlike the trigger counting problem (mentioned in the Introduction), here, most participants must know when the threshold is reached.

Lower Bounds for k-Set Agreement in Fault-Prone Networks

Pierre Fraigniaud
Inst. de Recherche en Informatique Fondamentale (IRIF), CNRS and Université Paris Cité, France

Minh Hang Nguyen
Inst. de Recherche en Informatique Fondamentale (IRIF), CNRS and Université Paris Cité, France

Ami Paz
Lab. Interdisciplinaire des Sciences du Numérique (LISN), CNRS & Université Paris-Saclay, France

Ulrich Schmid
TU Wien, Austria

Hugo Rincon-Galeana
TU Berlin, Germany

Abstract

We develop a new lower bound for k-set agreement in synchronous message-passing systems connected by an arbitrary directed communication network, where up to t processes may crash. Our result thus generalizes the $\lfloor t/k \rfloor + 1$ lower bound for complete networks in the t-resilient model by Chaudhuri, Herlihy, Lynch, and Tuttle [JACM 2000]. Moreover, it generalizes two lower bounds for oblivious algorithms in synchronous systems connected by an arbitrary undirected communication network known to the processes, namely, the domination number-based lower bound by Castañeda, Fraigniaud, Paz, Rajsbaum, Roy, and Travers [TCS 2021] for failure-free processes, and the radius-based lower bound in the t-resilient model by Fraigniaud, Nguyen, and Paz [STACS 2024].

Our topological proof non-trivially generalizes and extends the connectivity-based approach for the complete network, as presented in the book by Herlihy, Kozlov, and Rajsbaum (2013). It is based on a sequence of shellable carrier maps that, starting from a shellable input complex, determine the evolution of the protocol complex: During the first $\lfloor t/k \rfloor$ rounds, carrier maps that crash exactly k processes per round are used, which ensure high connectivity of their images. A Sperner's lemma style argument can thus be used to prove that k-set agreement is still impossible by that round. From round $\lfloor t/k \rfloor + 1$ up to our actual lower bound, a novel carrier map is employed, which maintains high connectivity. As a by-product, our proof also provides a strikingly simple lower-bound for k-set agreement in synchronous systems with an arbitrary communication network, where exactly $t \geq 0$ processes crash initially, i.e., before taking any step. We demonstrate that the resulting additional agreement overhead can be expressed via an appropriately defined radius of the communication graphs, and show that the usual input pseudosphere complex for k-set agreement can be replaced by an exponentially smaller input complex based on Kuhn triangulations, which we prove to be also shellable.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed computing, k-set agreement, time complexity, lower bounds, topology

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.31

Related Version *Full Version*: `http://arxiv.org/abs/2508.15562` [11]

Funding *Pierre Fraigniaud*: Additional support from ANR projects DUCAT (ANR-20-CE48-0006), ENEDISC (ANR-24-CE48-7768-01), and PREDICTIONS (ANR-23-CE48-0010), and from the InIDEX Project METALG.
Minh Hang Nguyen: Additional support from ANR projects DUCAT (ANR-20-CE48-0006), TEMPORAL (ANR-22-CE48-0001), ENEDISC (ANR-24-CE48-7768-01), and the European Union's



© Pierre Fraigniaud, Minh Hang Nguyen, Ami Paz, Ulrich Schmid, and Hugo Rincon-Galeana;
licensed under Creative Commons License CC-BY 4.0



Horizon 2020 program H2020-MSCA-COFUND-2019 Grant agreement n° 945332.
Hugo Rincon-Galeana: Supported by the German Research Foundation (DFG) SPP 2378 (project ReNO).

1 Introduction

In the k-set agreement task, introduced by Chaudhuri [7], each process starts with some input value belonging to an finite set of possible input values, and must irrevocably output a value usually referred to as its *decision* value. The output value decided by a process has to be the input of some process (strong validity condition), and, system-wide, no more that k different decision values may be decided (k-agreement condition). Whereas the case $k = 1$ (consensus) is well-understood, properly understanding k-set agreement for general $k > 1$ is notoriously difficult, even for simple computing models. Besides the inherent difficulty of handling a task that is less constrained than consensus, its analysis is considerably complicated by the fact that "classic" proof techniques are inadequate [1–3]. As a consequence, methods from combinatorial topology must usually be resorted to [12]. Such methods are very powerful, but often difficult to apply to concrete scenarios.

Unsurprisingly, these complications affect not only impossibility proofs for k-set agreement, but also termination-time lower bounds. In particular, in message-passing synchronous systems (where the processes communicate with each other in a sequence of synchronous, communication-closed *rounds* over some communication network), we are aware of only two substantially new results since the seminal tight $\lfloor t/k \rfloor + 1$ lower-bound established by Chaudhuri, Herlihy, Lynch, and Tuttle [8] for complete networks in the t-resilient model (where at most t processes may fail by crashing during any run).

The first one is the lower-bound by Castañeda, Fraigniaud, Paz, Rajsbaum, Roy, and Travers [5] (see also [10]) for *failure-free* processes connected by an arbitrary (connected) bidirectional communication network G that is commonly known to all nodes – this model is referred to as the KNOW-ALL model. It holds for *oblivious* algorithms only, that is, algorithms which exchange the sets of different input values seen so far using flooding-based communication, and only take decisions by these sets (and not, e.g., the time of a message arrival or the neighbor it arrived from). The lower bound essentially states that r rounds are necessary, where r is the smallest integer such that the graph $G_r = (V, E_r)$ obtained from $G = (V, E)$ by connecting by an edge every two nodes at distance at most r in G has domination number at most k.

The second one is the lower bound, established by Fraigniaud, Nguyen, and Paz [9], for *oblivious* algorithms in the t-resilient model with an arbitrary undirected communication network G. For $k = 1$, it essentially states that consensus requires r rounds, where $r = \text{radius}(G, t)$ is the *radius* of the network when up to t nodes may fail by crashing. Informally, the radius is defined as the minimum, taken over all nodes of the network, of the worst-case *finite* number of rounds required for broadcasting from a node over all possible failure patterns, hence can be defined via the *eccentricity* of certain nodes in G. The lower bound in [9] is tight for oblivious algorithms thanks to the algorithm in [6]. The consensus lower bound can be extended to k-set agreement using the same techniques as [10], but only if assuming a priori knowledge on the failure pattern.

In the current paper, we generalize the above results by developing a lower bound for the number of rounds for solving k-set agreement in the t-resilient model for arbitrary directed communication networks. It is fomulated via the "agreement overhead" caused by the presence of an arbitrary communication network over the mere case of the complete network.

▶ **Definition 1.1.** *Let G be a directed graph, and let $k \geq 1$ and $t \geq 0$ be integers. The agreement overhead* $\mathrm{ovh}(G, k, t)$ *is the smallest integer such that k-set agreement in G can be solved in $\lfloor t/k \rfloor + 1 + \mathrm{ovh}(G, k, t)$ rounds in the t-resilient model.*

The agreement overhead can hence be viewed as the penalty for not using the complete network but solely G. For the n-process complete network K_n, for every $k \geq 1$ and $t \geq 0$, $\mathrm{ovh}(K_n, k, t) = 0$, thanks to the lower bound established in [8].

1.1 Contributions

Our main lower bound result (Theorem 4.10) relies on two cornerstones:

(1) a proof that the $\lfloor t/k \rfloor + 1$ lower-bound for t-resilient systems over the complete communication network [8] also holds for every arbitrary network (which motivates the notion of agreement overhead), and

(2) a lower bound on the agreement overhead for an arbitrary communication graph $G = (V, E)$. For specifying the latter, recall that, for every $U \subseteq V$, $G[U]$ denotes the subgraph of G induced by the vertices in U. Given a set D of vertices, we denote by $\mathrm{ecc}(D, G)$ the eccentricity of D in G, i.e., the number of rounds D need to collectively broadcast to all the graph's nodes. We then define the (t, k)-radius of G as

$$\mathrm{rad}(G, t, k) = \min_{D \subseteq V, |D| = t+k} \max_{D' \subseteq D, |D'| = t} \mathrm{ecc}(D \setminus D', G[V \setminus D']) \tag{1}$$

and show that the agreement overhead satisfies $\mathrm{ovh}(G, k, t) \geq \mathrm{rad}(G, k, t) - 1$.

Consequently, any algorithm solving k-set agreement in G under the t-resilient model must perform at least $\lfloor \frac{t}{k} \rfloor + \mathrm{rad}(G, t, k)$ rounds (Theorem 4.10). For the special case of $t = 0$, our lower bound is equivalent to the one established in [5] for the KNOW-ALL model. For $t > 0$, our lower bound on the agreement overhead also gives a lower bound of $\mathrm{rad}(G, k, t)$ for solving k-set agreement in arbitrary networks with t initially dead processes (Theorem 4.11).

Our paper also advances the state of the art of topological modeling as follows:

(3) We introduce a novel carrier map that governs the evolution of a shellable protocol complex in systems connected by an arbitrary directed communication graph G (that may even vary from round to round) with t initially dead processes, and show that it maintains high connectivity during $\mathrm{ovh}(G, k, t)$ rounds. For $t = 0$, our carrier map allows a much simpler analysis of the setting studied in [5]. For $t > 0$, we also demonstrate how to generalize the scissors cut-based analysis in [5] for handling the case $t > 0$ as well, and show that the resulting lower bound is equivalent to the one $\mathrm{ovh}(G, k, t) + 1$ established by our approach.

(4) We non-trivially generalize, extend and also correct the topological proof technique for the $\lfloor t/k \rfloor + 1$ lower bound in complete networks sketched in [12, Ch. 13] to arbitrary directed communication graphs (that may also vary from round to round). Our approach starts out from a shellable input complex, and utilizes a sequence of shellable carrier maps that crash exactly k processes per round for modeling the evolution of the protocol complex. Since these carrier maps maintain high connectivity during the first $\lfloor t/k \rfloor$ rounds, a Sperner-lemma style argument can be used to prove that k-set agreement is still impossible. Our contribution not only adds details missing in [12, Ch. 13] (e.g., the strictness proof of the carrier maps, and the Sperner-style argument), but also fixes a non-trivial error by replacing the rigidity requirement for the carrier maps (which does not hold) by a novel, weaker condition.

(5) We prove that the Kuhn triangulation [8], which is exponentially smaller than the standard pseudosphere complex used as the input complex for k-set agreement in [12, Ch. 13], is shellable. We can hence seamlessly replace the pseudosphere input complex in our analysis by Kuhn triangulations.

Whereas the focus of our results are lower bounds, the question of tightness obviously arises. So far, we do not know whether and for which choices of G, k and t our lower bound in Theorem 4.10 is tight. We must hence leave this question to future research. We nevertheless include the following upper-bound result:

(6) We present an upper bound on the agreement overhead by generalizing the algorithm for the clique K_n in [8] to an arbitrary communication network $G = (V, E)$. For $S \subseteq V$, let $G[V \setminus S]$ denote the subgraph of G induced by the nodes in $V \setminus S$, and let $D(G, t) = \max_{S \subseteq V, |S| \leq t} \operatorname{diam}(G[V \setminus S])$, where diam denotes the diameter. By following the arguments in [8], we show that there exists an algorithm solving k-set agreement in G in $\lfloor \frac{t}{k} \rfloor + D(G, t)$ rounds. As a consequence, $\operatorname{ovh}(G, k, t) \leq D(G, t) - 1$.

Paper organization.[1] In Section 2, we introduce our system model, and the basics of the topological modeling. In Section 3, we revisit the topological round-by-round connectivity analysis of [12, Ch. 13]. Section 4 provides our lower bounds on the agreement overhead, and Section 5 provides our upper-bound result. Some conclusions in Section 6 complete our paper. All our proofs are provided in the appendix.

2 System Model

2.1 Computational Model

Our computational model is similar to the one used by Fraigniaud, Nguyen and Paz [9], albeit we consider full-history protocols, and general (i.e., non-necessarily oblivious) algorithms. We consider a finite set of n processes with names $\Pi = \{p_1, \ldots, p_n\}$, that are ordered according to their index set $[n] = 1, \ldots, n$. Processes communicate in lock-step synchronous rounds via point-to-point directed links, that is, any message that is sent during a round r will be received in the same round r, and we do not consider the possibility that messages arrive at later rounds. We consider that all processes start simultaneously at round 1.

We assume that processes are represented by deterministic state machines, and have a well-defined local state that also includes the complete history of received messages. Thus, we consider a protocol to be defined by state transitions as well as a communication function and a decision function. In this paper, we will not focus on the protocol specifications, since it is fairly simple to derive them from the particular protocols that we consider. Instead, for the sake of readability and succinctness, we will sketch the protocols by specifying the messages that a process is able to send at each round, the information captured by the local states, and whether or not a process is ready to decide on an output value.

In a given round, processes can communicate using a fixed network topology, represented by a directed *communication graph* $G = (V, E)$, where $V = \Pi$. That is, a process p can send a message directly to another process q if and only if $(p, q) \in E$. We assume that E contains all self-loops $\{(p, p) \mid p \in V\}$. The processes are aware of the communication graph G. A

[1] Lack of space does not allow us to include all our results, which can be found in the full version [11] of our paper.

process p is able to send a message to any other process in its set of out-neighbors in G, denoted by $\mathrm{Out}_p(G) := \{q \in V \mid (p,q) \in E\}$. Symmetrically, p can only receive a message from a process in its set of in-neighbors in G, denoted by $\mathrm{In}_p(G) := \{q \in V \mid (q,p) \in E(G)\}$.

We consider the t-resilient model, where up to t processes may permanently crash in every execution, in any round. Crashes may be unclean, thus a process p may still send a message to a non-empty subset of $\mathrm{Out}_p(G)$ before crashing. The set of faulty processes crashing in a given execution is denoted as F with $|F| \leq t$, and is arbitrary and unknown to the processes.

For the sake of completeness, we also provide a formal description of the k-set agreement problem, which constitutes the main focus of this paper. Every process p_i has a local *input value* x_i taken arbitrarily from a finite set $\mathcal{V}$ with $|\mathcal{V}| \geq k+1$, which is often assumed to be just $\mathcal{V} = [k+1] = \{1, \ldots, k+1\}$. Every correct process p_i must irrevocably assign some decision value to a local *output variable* y_i eventually, which is initialized to $y_i = \perp$ with $\perp \notin \mathcal{V}$. In essence, k-set agreement is a relaxed instance of consensus, in which the agreement condition is relaxed to accept at most k different decision values. More precisely, k-set agreement is defined by the following conditions:

- **Strong Validity:** If a process p_i decides output value y_i, then y_i is the input value x_j of a process p_j
- ***k*-Agreement:** In every execution, if $\mathcal{O}$ denotes the set of all decision values of the processes that decide in that execution, then $|\mathcal{O}| \leq k$.
- **Termination:** Every non-faulty process $p_i \notin F$ must eventually and irrevocably decide on some value $y_i \neq \perp$.

2.2 Basics of combinatorial topology

Our analysis of k-set agreement relies on combinatorial topology [12]. Most notably, we develop novel topological techniques that allow us to ensure high-order connectivity, which seamlessly translates to a lower bound for k-set agreement. We now provide some basic definitions on simplicial complexes, which will be used heavily in the paper.

Intuitively, simplicial complexes may be thought of as a "higher dimensional" instance of an undirected graph. Indeed, in addition to vertices and edges, a simplicial complex may have faces of higher dimension.

▶ **Definition 2.1** (Simplicial Complex). *A pair $\mathcal{K} = (V(\mathcal{K}), F(\mathcal{K}))$, where $V(\mathcal{K})$ is a set, and $F(\mathcal{K}) \subseteq 2^{V(\mathcal{K})} \setminus \varnothing$ is a collection of subsets of $V(\mathcal{K})$ is a simplicial complex if, for any $\sigma \in F(\mathcal{K})$, and any $\sigma' \subseteq \sigma$, $\sigma' \in F(\mathcal{K})$. $V(\mathcal{K})$ is called the vertex set, and $F(\mathcal{K})$ the set of faces called simplices (singular: simplex). For notational simplicity, we will occasionally refer to simplicial complexes as* complexes.

Note that, following the convention in [12], we will very rarely (cf. Definition 3.4) also consider the empty "simplex" $\varnothing$.

The maximal faces (by inclusion) of a simplicial complex are called *facets*. Since faces are downward closed, then the facets are sufficient for fully determining a simplicial complex. The dimension of a face σ is defined as $\dim(\sigma) = |\sigma| - 1$. The dimension of a simplicial complex $\mathcal{K} = (V(\mathcal{K}), F(\mathcal{K}))$ is defined as $\max_{\sigma \in F(\mathcal{K})} \dim(\sigma)$. A complex is *pure* if all of its facets have the same dimension, and a complex is *impure* if it is not pure. For a face σ of a pure complex with facet dimension d, we denote by $\mathrm{codim}(\sigma) = d - \dim(\sigma)$ the *co-dimension* of σ, and by $\mathrm{Face}_k\, \sigma = \{\rho \mid \rho \subseteq \sigma \text{ with } \dim(\rho) = k\}$ the set of all k-faces of σ.

For any two simplicial complexes $\mathcal{K}$ and $\mathcal{L}$, $\mathcal{L}$ is a *subcomplex* of $\mathcal{K}$, denoted by $\mathcal{L} \subseteq \mathcal{K}$, if $V(\mathcal{L}) \subseteq V(\mathcal{K})$ and $F(\mathcal{L}) \subseteq F(\mathcal{K})$. For any $d \geq 0$, the d-skeleton $\text{skel}_d(\mathcal{K})$ is the subcomplex of $\mathcal{K}$ consisting of all simplices of dimension at most d.

The morphisms (i.e., structure-preserving maps) for simplicial complexes are called *simplicial* maps:

▶ **Definition 2.2** (Simplicial map). *Let $\mathcal{K}$ and $\mathcal{L}$ be simplicial complexes. A mapping $\mu : V(\mathcal{K}) \to V(\mathcal{L})$ is a simplicial map if, for every $\sigma \in F(\mathcal{K})$, $\mu(\sigma) \in F(\mathcal{L})$.*

For our analysis, we also need to consider other maps beyond simplicial maps. Since we are interested in the evolution of configurations of processes, which are represented via faces of a simplicial complex, we need to consider functions that map individual simplices to sets of simplices.

▶ **Definition 2.3** (Carrier maps). *Let $\mathcal{K}$ and $\mathcal{L}$ be simplicial complexes, and $\Psi : F(\mathcal{K}) \to 2^{F(\mathcal{L})}$ be a function that maps faces of $\mathcal{K}$ into sets of faces of $\mathcal{L}$ such that, for every simplex $\sigma \in \mathcal{K}$, $\Psi(\sigma)$ is a subcomplex of $\mathcal{L}$. Ψ is a* carrier map *if, for every two simplices σ and κ in $F(\mathcal{K})$, $\Psi(\sigma \cap \kappa) \subseteq \Psi(\sigma) \cap \Psi(\kappa)$. Moreover,*

- *Ψ is* strict *if $\Psi(\sigma \cap \kappa) = \Psi(\sigma) \cap \Psi(\kappa)$, and*
- *Ψ is* rigid *if, for every $\sigma \in F(\mathcal{K})$, $\Psi(\sigma)$ is pure and of dimension $\dim(\sigma)$.*

Note that the definition above also allows $\Psi(\sigma) = \emptyset$, the empty complex.

In addition to the vertices and faces, a simplicial complex $\mathcal{K}$ may be endowed with a vertex coloring $\chi : V(\mathcal{K}) \to \mathcal{C}$, where $\mathcal{C}$ is the *color set*. We say that a vertex coloring χ is proper on $\mathcal{K}$ if for any simplex $\sigma \in F(\mathcal{K})$, the restriction $\chi_{|\sigma}$ of χ on σ is injective. We say that a pair $\mathcal{K}_\chi := (\mathcal{K}, \chi)$ is a *chromatic simplicial complex* if $\mathcal{K}$ is a simplicial complex, and $\chi : V(\mathcal{K}) \to \mathcal{C}$ is a proper vertex coloring. (As we shall see in the next section, in the context of using complexes to model distributed computing, the color of a vertex is merely a process ID.) Let $\mathcal{K}_\chi := (\mathcal{K}, \chi)$ and $\mathcal{L}_{\chi'} := (\mathcal{L}, \chi')$ be chromatic simplicial complexes. A simplicial map $\mu : V(\mathcal{K}) \to V(\mathcal{L})$ is a *chromatic map* if, for every $v \in V(\mathcal{K})$, $\chi(v) = \chi'(\mu(v))$, i.e., μ is color-preserving. For notational simplicity, when it is clear from the context, we will omit mentioning the vertex coloring explicitly.

2.3 Topological modeling of distributed systems

Simplicial complexes are particularly useful for representing system configurations, both for inputs and outputs, and for describing mid-run states. Vertices are used for representing local states, while faces represent (partial) configurations.

The *input complex* $\mathcal{I}$ is used for representing all possible initial configurations. Its vertices (p_i, x_i) consist of a process name $p_i = \chi((p_i, x_i)) \in \Pi$ that is used as its color, and some input value $x_i \in \mathcal{V}$. A facet σ of the input complex consists of n vertices $v_1, \ldots, v_n$, with $v_i = (p_i, x_i)$ for $1 \leq i \leq n$ that represent some initial configuration.

The *output complex* $\mathcal{O}$ is used for representing all possible decision configurations. Its vertices (p_i, y_i) consist of a process name $p_i = \chi((p_i, y_i)) \in \Pi$ that is used as its color, and some output value $y_i \in \mathcal{V}$.

The *protocol complex* $\mathcal{P}^r$ at the end of round $r \geq 1$ is used for representing all possible system configurations after r rounds of execution. Its vertices (p_i, λ_i) consist of a process name $p_i = \chi((p_i, \lambda_i)) \in \Pi$ that is used as its color, and the local state λ_i of p_i at the end of round r. Since processes can crash in the t-resilient model, the protocol complex may not be pure. A facet σ of the protocol complex consists of $n' \geq n - t$ vertices $v_{\pi(1)}, \ldots, v_{\pi(n')}$, with $v_{\pi(i)} = (p_{\pi(i)}, \lambda_{\pi(i)})$ for $1 \leq i \leq n'$ that represent some possible system configuration after r rounds. We set $\mathcal{P}^0 = \mathcal{I}$.

For any of the simplicial complexes above, $\mathsf{names}(\sigma)$ denotes the set of process names corresponding to the vertices of a face σ, i.e., $\mathsf{names}(\sigma) = \chi(\sigma)$.

In topological modeling, problems like k-set agreement are specified as a *task* $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$, where $\Delta : \mathcal{I} \to \mathcal{O}$ is a carrier map that specifies the allowed decision configurations $\Delta(\sigma)$ for a given face $\sigma \in \mathcal{I}$.

▶ **Definition 2.4** (Task solvability). *A task $\mathcal{T}$ is solvable with respect to a protocol complex $\mathcal{P}$ if there exists a simplicial chromatic map $\delta : \mathcal{P} \to \mathcal{O}$ that agrees with Δ, that is, for every $\sigma \in \mathcal{I}$, and for every $\kappa \in \mathcal{P}_\sigma$, $\delta(\kappa) \in \Delta(\sigma)$, where $\mathcal{P}_\sigma$ is the collection of faces of $\mathcal{P}$ reachable when only processes in σ run, with inputs taken from σ.*

Note that, for each face $\sigma \in \mathcal{I}$, $\mathcal{P}_\sigma$ may or may not be empty depending on the executions allowed by the underlying model. For instance, for wait-free computing in the IIS model, $\mathcal{P}_\sigma \neq \varnothing$ for every σ because $|\sigma| \geq 1$ and up to all but one processes can crash. Instead, for synchronous *failure-free* shared-memory computing, $\mathcal{P}_\sigma \neq \varnothing$ if and only if $|\sigma| = n$, whereas for t-resilient synchronous message-passing, $\mathcal{P}_\sigma \neq \varnothing$ if and only if $|\sigma| \geq n - t$.

3 Connectivity-Based Topological Analysis of Synchronous Systems

In this section, we re-visit the round-by-round topological analysis of the lower-bound for deterministic k-set agreement algorithms in the synchronous t-resilient model for the complete graph in [12, Ch. 13]. In a nutshell, this analysis shows that too few rounds of communication lead to a protocol complex that is too highly connected for allowing the existence of a simplicial chromatic map to the output complex of k-set agreement. Since some parts of the original proof are not entirely correct or have been omitted, we revisit these parts in Section 3.2 after presenting some basic ingredients for the proof. In [11, Sec. 3.3], we generalize the analysis for the complete graph to arbitrary communication graphs, and prove formally that the lower bound $\lfloor t/k \rfloor + 1$ for complete graphs also applies to arbitrary communication graphs.

Note that [12, Ch. 13] assumes a system consisting of $n+1$ processes named $\{P_0, \ldots, P_n\}$, with index set $\{0, \ldots, n\}$, whereas our system model considers n processes named $\{p_1, \ldots, p_n\}$, with index set $\{1, \ldots, n\}$. For uniformity, we decided to stick to the latter notation, which makes it necessary to "translate" the original and revised definitions and lemmas of [12, Ch. 13]. In a nutshell, this primarily requires replacing n occurring in the dimension of a face by $n-1$.

3.1 Basic ingredients

In this subsection, we introduce the key ingredients needed for the topological analysis in [12], which tracks the connectivity properties of the sequence of protocol complexes over successive rounds. We start out with the definition of a pseudosphere, a particular type of simplicial complexes. As we shall see, the input complex $\mathcal{I}$ of k-set agreement is a pseudosphere.

▶ **Definition 3.1** (Pseudosphere [12, Def. 13.3.1]). *Let $\varnothing \neq I \subseteq [n]$ be a finite index set. For each $i \in I$, let p_i be a process name, indexed such that if $i \neq j$, then $p_i \neq p_j$, and let V_i be a non-empty set. The* pseudosphere *complex $\Psi(\{(p_i, V_i) \mid i \in I\})$ is defined as follows:*

- *Every pair (p_i, v) with $i \in I$ and $v \in V_i$ is a vertex., and*
- *for every index set $J \subseteq I$, any set $\{(p_j, v_j) \mid j \in J\}$ such that $v_j \in V_j$ for all $j \in J$ is a simplex.*

Note that, for a given simplex σ, and a given set of values $\mathcal{V}$, we sometimes use the shorthand $\Psi(\sigma, \mathcal{V})$ to denote $\Psi(\{(p, \mathcal{V}) \mid p \in \mathsf{names}(\sigma)\})$, where the processes in $\mathsf{names}(\sigma)$ define the corresponding index set I. An essential feature of the protocol complexes arising in the round-by-round connectivity analysis is that they are *shellable*. Intuitively, a pure d-dimensional complex is shellable if it can be built by gluing together its facets in some specific order, called *shelling order*, such that a newly added facet intersects the already built subcomplex in $(d-1)$-dimensional faces only.

▶ **Definition 3.2** (Shellable complex). *A simplicial complex $\mathcal{K}$ is* shellable *if it is pure, of dimension d for some $d \geq 0$, and its facets can be arranged in a linear order $\phi_0, \ldots, \phi_N$, called a* shelling order*, in such a way that, for every $k \in \{1, \ldots, N\}$, the subcomplex $\left(\bigcup_{i=0}^{k-1} \phi_i\right) \cap \phi_k$ is the union of $(d-1)$-dimensional faces of ϕ_k.*

The following alternative definition of shellability is easier to use in proofs.

▶ **Definition 3.3** (Shellability properties [12, Fact 13.1.3]). *An order $\phi_0, \ldots, \phi_N$ of the facets of a pure complex $\mathcal{K}$ is a shelling order if and only if, for any two facets ϕ_a and ϕ_b with $a < b$ in that order, there exists ϕ_c with $c < b$ such that (i) $\phi_a \cap \phi_b \subseteq \phi_c \cap \phi_b$, and (ii) $|\phi_b \setminus \phi_c| = 1$.*

Note that the face ϕ_c guaranteed by Definition 3.3 can depend on ϕ_a. Moreover, ϕ_c is usually not unique, i.e., there might be several choices all satisfying the above properties.

It was shown in [12, Lem. 13.2.2] (resp., [12, Thm. 13.3.6]) that the d-skeleton, for any dimension $d \geq 0$, of any simplex (resp., of any pseudosphere) is shellable. The proofs of these facts are based on the following orderings.

▶ **Definition 3.4** (Face order, adapted from [12, Def. 13.2.1]). *Let $\sigma = \{v_1, \ldots, v_n\}$ be an $(n-1)$-simplex, together with a total ordering on its vertices $v_1, \ldots, v_n$ given by index order. Each face τ of σ has an associated* signature*, denoted by $\mathrm{sig}(\tau)$, defined as the Boolean string $(\mathrm{sig}(\tau)[1], \ldots, \mathrm{sig}(\tau)[n])$ of length n whose i-th entry is*

$$\mathrm{sig}(\tau)[i] = \begin{cases} \bot & \text{if } v_i \in \tau, \\ \top & \text{if } v_i \notin \tau. \end{cases} \tag{2}$$

The face order $<_f$ of two faces τ_1, τ_2 of σ is defined as $\tau_1 <_f \tau_2$ if $\mathrm{sig}(\tau_1) <_{lex} \mathrm{sig}(\tau_2)$, i.e., $\mathrm{sig}(\tau_1)$ is lexicographically smaller than $\mathrm{sig}(\tau_2)$, where $\bot < \top$.

Note that the empty "simplex" $\varnothing$ is the smallest in the face order of Definition 3.4, and $\mathrm{sig}(\{v_1, \ldots, v_n\})$ is the largest. Definition 3.4 has been slightly adapted from [12, Sec. 13.2.1], by using the more precise notation $\mathrm{sig}(\tau)$ instead of just τ. Informally, $\mathrm{sig}(\tau)$ just encodes the set of processes involved in (the vertices of) a simplex τ. The face order in Definition 3.4 can be used to show that, for any dimension $d \geq 0$, the d-skeleton, of any simplex is shellable.

▶ **Definition 3.5** (Pseudosphere order [12, Def. 13.3.5]). *Let $\phi_a = \{(p_i, \lambda_i) \mid i \in [n]\}$ and $\phi_b = \{(p_i, \mu_i) \mid i \in [n]\}$ be two facets of a pseudosphere $\Psi(\{(p_i, V_i) \mid i \in [n]\})$ where each V_i is an ordered set. The order relation $<_p$ orders these facets lexicographically by value, i.e., $\phi_a <_p \phi_b$ if there exists $\ell \in [n]$ such that $\lambda_i = \mu_i$ for every $1 \leq i < \ell$, and $\lambda_\ell < \mu_\ell$.*

It is known that every skeleton of a shellable complex is shellable [4] (see [11, Thm. 3.6] for the detailed proof):

▶ **Theorem 3.6** (Shellability of skeletons of pseudospheres). *Let $\Psi = \Psi(\{(p_i, V_i) \mid i \in [n]\})$ be an $(n-1)$-dimensional pseudosphere. For every $d \in \{0, \ldots, n-1\}$, the d-skeleton of Ψ is shellable via the shelling order $<$ defined as $\phi_a < \phi_b \iff (\phi_a <_f \phi_b) \vee \big((\mathrm{sig}(\phi_a) = \mathrm{sig}(\phi_b)) \wedge (\phi_a <_p \phi_b)\big)$.*

As our last basic ingredient, we provide a proof of the well-known but often quite informally argued fact that k-set agreement is impossible if the protocol complex is too highly connected (see, e.g., [12, Thm. 10.3.1]). Informally, a complex $\mathcal{K}$ is k-connected, if it does not contain a "hole" of dimension k or lower. For $k = 0$, which captures the consensus impossibility, for example, $\mathcal{K}$ must not be (path-)connected, i.e., 0-connected. More generally, 1-connectivity refers to the ability to contract 1-dimensional loops, 2-connectivity refers to the ability to contract 2-dimensional spheres, etc.

Let $\mathcal{T} = (\mathcal{I}, \mathcal{O}, \Delta)$ be the k-set agreement task as specified in Section 2.3. In particular, $\mathcal{I} = \Psi(\{(p_i, [k+1]) \mid i \in [n]\})$ is a pseudosphere.

▶ **Definition 3.7.** *For every $J \subseteq [k+1]$, we define $\mathcal{P}[J]$ as the minimal complex including $\mathcal{P}_\sigma$ for all $\sigma \in \Psi(\{(p_i, J) \mid i \in [n]\})$.*

▶ **Theorem 3.8.** *If $\mathcal{P}[J]$ is $(\dim(J) - 1)$-connected for all $J \subseteq [k+1]$, then k-set agreement is not solvable with respect to $\mathcal{P}$.*

3.2 The round-by-round connectivity analysis of [12] revisited

During our attempts to generalize the round-by-round topological modeling and analysis of [12, Ch. 13] for the complete graph to arbitrary networks, we figured out that the original analysis in [12] is not entirely correct. More specifically, the analysis there assumes that the involved carrier maps are rigid, which cannot be guaranteed in the executions considered for synchronous k-set agreement where exactly k processes crash per round. We hence provide here a revised analysis for the case of the complete graph (i.e., the clique of our n processes), where rigidity is replaced by a weaker condition (Definition 3.9 below). Unfortunately, this change forces us to re-phrase and re-prove most of the lemmas of the original analysis. Moreover, we have to add a non-trivial strictness proof in Lemma 3.14 below, which was lacking in [12, Ch. 13]. The next definition relaxes Definition 13.4.1 in [12] by removing the rigidity condition.

▶ **Definition 3.9** (q-connected carrier map)**.** *Let $q \geq 0$ be an integer, and let $\mathcal{L}$ and $\mathcal{K}$ be simplicial complexes, where $\mathcal{K}$ is pure. A carrier map $f : \mathcal{K} \to 2^{\mathcal{L}}$ is q-connected if it is strict, and, for every $\sigma \in \mathcal{K}$, the simplicial complex $f(\sigma)$ is $(q - \operatorname{codim}(\sigma))$-connected.*

The following Lemma 3.10 is exactly the same as [12, Lem. 13.4.2]. Indeed, thanks to our new Definition 3.9 of a q-connected carrier map, the original proof holds literally as well, as faces with $\operatorname{codim}(\sigma) > q$ are not considered anyway.

▶ **Lemma 3.10** ([12, Lem. 13.4.2])**.** *For every integer $q \geq 0$, if $\mathcal{K}$ is a pure shellable simplicial complex, and $f : \mathcal{K} \to 2^{\mathcal{L}}$ is a q-connected carrier map, then the simplicial complex $f(\mathcal{K})$ is q-connected.*

In Definition 13.4.3 in [12], a shellable carrier map $f : \mathcal{K} \to 2^{\mathcal{L}}$ was defined as a *rigid* and strict carrier map such that, for each $\sigma \in \mathcal{K}$, the complex $f(\sigma)$ is shellable. Since the carrier maps we study later on are not rigid, we need to weaken this definition as follows.

▶ **Definition 3.11** (q-shellable carrier map)**.** *Let $q \geq 0$ be an integer. A carrier map $f : \mathcal{K} \to 2^{\mathcal{L}}$ is q-shellable if it is strict, and, for each $\sigma \in \mathcal{K}$ satisfying $\operatorname{codim}(\sigma) \leq q + 1$, the complex $f(\sigma)$ is shellable (and hence pure).*

Note carefully that a q-shellable carrier map only guarantees that the image $f(\sigma)$ of a given *single* face $\sigma \in \mathcal{K}$ is shellable. To set the stage for extending the analysis of this section for arbitrary communication graphs later on, we introduce some notation already

here: For a given simplex $\kappa \in \mathcal{K}_{i-1}$ (that will become clear from context), we abbreviate $\overline{\mathcal{K}_i} := f_{i-1}(\kappa) \subseteq \mathcal{K}_i$ and restrict our attention only to (facets of) this subcomplex when proving shellability. Indeed, this will become necessary, since one cannot infer that the entire complex $\mathcal{K}_i$ is also shellable, as we prove in [11, Sec. 3.4].

The following is an appropriately refined version of Lemma 13.4.4 in [12].

▶ **Lemma 3.12.** *Let $q \geq 0$ be an integer, and let us consider a sequence of pure complexes, and carrier maps $\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \mathcal{K}_2$, where f_0 is a q-shellable carrier map, f_1 is a q-connected carrier map, and, for every $\sigma \in \mathcal{K}_0$ with $\operatorname{codim}(\sigma) \leq q+1$, $\operatorname{codim}(\sigma) \geq \operatorname{codim}(f_0(\sigma))$. Then, $f_1 \circ f_0$ is a q-connected carrier map.*

Similarly, we need a refined version of Lemma 13.4.5 in [12].

▶ **Lemma 3.13.** *Let $q \geq 0$ and $\ell \geq 0$ be integers, and let us consider a sequence of pure complexes, and carrier maps $\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \ldots \xrightarrow{f_\ell} \mathcal{K}_{\ell+1}$, such that the carrier maps $f_0, \ldots, f_{\ell-1}$ are q-shellable, the carrier map f_ℓ is q-connected, and, for every $i \in \{0, \ldots, \ell-1\}$, and every $\sigma \in \mathcal{K}_i$ with $\operatorname{codim}(\sigma) \leq q+1$, $\operatorname{codim}(\sigma) \geq \operatorname{codim}(f_i(\sigma))$. Then, $f_\ell \circ \cdots \circ f_0$ is a q-connected carrier map.*

We want to prove a variant of [12, Thm. 13.5.7] adapted to our refined modeling. For some N to be determined later, consider a sequence

$$\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{N-1}} \mathcal{K}_N \xrightarrow{id} \mathcal{K}_N \tag{3}$$

where $\mathcal{K}_0$ is the (shellable) input complex for k-set agreement, each $\mathcal{K}_i$ is the image of $\mathcal{K}_{i-1}$ under f_{i-1} (i.e., f_{i-1} is surjective), and

$$f_i(\sigma) = \bigcup_{\tau \in \operatorname{Face}_{n-1-k(i+1)} \sigma} \Psi(\tau, [\tau, \sigma]), \tag{4}$$

where $\Psi(\tau, [\tau, \sigma])$ denotes the pseudosphere obtained by independently labeling the processes in $\mathsf{names}(\tau)$ with one of the simplices in $\{\rho \mid \tau \subseteq \rho \subseteq \sigma\}$. Intuitively, a vertex (p_j, ρ) represents the situation where p_j receives exactly the information in ρ in round $i+1$. As in [12, Sec. 13.5.2], the carrier map f_i applied to σ is the execution map representing round $i+1$ starting from some face $\sigma \in \mathcal{I}$, where the communication graph is a clique, and exactly k additional processes crash in round $i+1$ (i.e., a total of $i \cdot k$ processes have already failed during the i previous rounds).

▶ **Lemma 3.14.** *For every $i \in \{0, \ldots, N-1\}$, f_i is $(k-1)$-shellable, and, for every $\sigma \in \mathcal{K}_i$ with $\operatorname{codim}(\sigma) \leq k$, $\operatorname{codim}(f_i(\sigma)) \leq \operatorname{codim}(\sigma)$.*

Since the identity map id in Equation (3) is trivially a q-connected carrier map satisfying Definition 3.9, we can apply Lemma 3.13 for $N = \lfloor t/k \rfloor$ equal to the maximum number of rounds where k processes can crash to immediately get:

▶ **Lemma 3.15.** *For $N = \lfloor t/k \rfloor$, $f_{N-1} \circ \cdots \circ f_0 : \mathcal{K}_0 \to \mathcal{K}_N$ is a $(k-1)$-connected carrier map.*

▶ **Theorem 3.16.** *For integers $t \geq 0$ and $k \geq 1$, let $\mathcal{P}$ be the protocol complex of k-set agreement after $N = \lfloor \frac{t}{k} \rfloor$ rounds in the synchronous t-resilient model with the complete communication graph, where exactly k processes crash per round. For every $J \subseteq [k+1]$, $\mathcal{P}[J]$ is a $(\dim(J)-1)$-connected subcomplex.*

By combining Theorems 3.8 and 3.16, we finally get the well-known lower bound:

▶ **Corollary 3.17.** *The k-set agreement task cannot be solved in less than $\lfloor \frac{t}{k} \rfloor + 1$ rounds in the synchronous t-resilient model.*

4 A Lower Bound for the Agreement Overhead for Arbitrary Graphs

In this section, we will derive a lower bound for the agreement overhead (Definition 1.1), i.e., the number of additional rounds necessary for solving k-set agreement in t-resilient systems after the first $N = \lfloor t/k \rfloor$ crashing rounds. Interestingly, this can be done via two substantially different approaches, which will be presented below and in [11, Sec. 4.2]. Moreover, as a byproduct of our analyses, we will also establish a lower bound for k-set agreement in systems with t initially dead processes connected by arbitrary communication graphs. For simplicity, we henceforth assume that k evenly divides t, and no further crashes occur after round $N = t/k$. If this is not the case, the missing $t - k\lfloor t/k \rfloor$ crashes could only increase our lower bound, which would further complicate our analysis, and so we discard this option here.

In Definition 4.1 below, we will introduce a novel carrier map g that captures the agreement overhead caused by *arbitrary communication graphs*, beyond the mere case of the clique. This carrier map has been inspired by the *scissors cuts* introduced in [5], which were used to prove a lower bound for solving k-set agreement with oblivious algorithms in the KNOW-ALL model (which is *failure-free*). Our approach however differs from the original scissors cuts in several important ways. First, we admit directed graphs and general full-information algorithms, and replace the original pseudosphere input complex by the *source complex* $\mathcal{P}_N$, which is one of the following two cases:

- $\mathcal{P}_N$ is the (locally shellable) complex $\mathcal{K}_N = \mathcal{P}^{(N)}$ for arbitrary graphs (formally introduced in [11, Sec. 3.3]). This will allow us to paste together the agreement overhead lower bound determined in this section with the $\lfloor t/k \rfloor$ lower bound caused by process crashes.
- $\mathcal{P}_N$ is the (shellable) $(n-t-1)$-skeleton $\mathrm{skel}_{n-t-1}\big(\Psi(\{(p_i, [k+1]) \mid i \in [n]\})\big)$ of the pseudosphere given in Theorem 3.6, which we subsequently abbreviate as $\Psi(n, k+1)$ for conciseness, which contains all the faces of the full pseudosphere $\Psi(\{(p_i, [k+1]) \mid i \in [n]\})$ with at most $n-t$ vertices. Note that actually $N = 0$ in this case, albeit we will not make this explicit later on, but just stick to $\mathcal{P}_N$ to denote the source complex for uniformity. This will result in a lower bound (Theorem 4.11) for k-set agreement with t initially dead processes. Note that the total number of processes that appear in *all* the facets of $\Psi(n, k+1)$ together is n here.

The second main difference w.r.t. [5] is that g will be a proper carrier map, which also specifies the images of arbitrary faces of the source complex, and not only images of facets. It is particularly simple, however, since it resembles a (non-rigid) simplicial map in that $g(\sigma)$ returns the subcomplex corresponding to a *single* simplex ρ, or else $g(\sigma) = \varnothing$. In order not to unnecessarily clutter our notation, we will hence subsequently pretend as if $g(\sigma)$ only consisted of a single simplex ρ or $\varnothing$ only. One of the particularly appealing consequences of g's simple image is that it allows us to replace the very complex topological analysis in [5] by a strikingly simple connectivity argument.

Generally, we assume that $g : \mathcal{P}_N \to \mathcal{P}_M$ given in Definition 4.1 models the failure-free execution in rounds $N+1, N+2, \ldots, M$ for some $M > N$, where $M-N$ will finally determine our desired agreement overhead lower bound. Consider the execution starting in some facet $\phi \in \mathcal{P}_N$, which involves exactly $n-t$ processes with index set $I_\phi \subseteq [n]$ and consists of vertices of the form (p_i, λ_i), $i \in I_\phi$, each with a process name $p_i = \mathsf{names}((p_i, \lambda_i)) \in \Pi$ used as its color, and a label λ_i (which denotes p_i's local view at the end of round N), consisting of

- the vertices of the processes that managed to successfully send to p_i up to round N, if $N > 0$, or
- p_i's initial value x_i if $N = 0$, i.e., when the source complex is $\mathrm{skel}_{n-t-1}\big(\Psi(n, k+1)\big)$.

In either case, λ_i encodes the complete heard-of history of p_i up to round N, due to the fact that we are assuming full-information protocols.

We assume that $G_{N+1}, \ldots, G_M$ is the sequence of communication graphs governing rounds $N+1, \ldots, M$. These graphs may be different and known to the processes; clearly, assuming a static graph $G = G_{N+1} = \ldots G_M$ as in Section 1 can make our lower bound only stronger. We will abbreviate this sequence by G for brevity, and define G_ϕ to be the product $G_{N+1,\phi} \circ G_{N+2,\phi} \circ \cdots \circ G_{M,\phi}$, where $G_{N+1,\phi}, \ldots, G_{M,\phi}$ are the graphs induced by the nodes $\mathsf{names}(\phi)$ on the graphs in the sequence G.

It is worth mentioning that our carrier map g actually focuses on a subset of all the possible executions, as it is sufficient for a lower bound. Namely, g considers the case where processes that crashed in round N did crash *cleanly* only (they failed to send messages to all their neighbors). Note that this somehow resembles the situation of the carrier maps f_i used in Section 3, which also only covered a submodel of all possible executions, namely, the one where exactly k processes crash per round. The map g accomplishes this by "discarding" executions starting from facets in $\mathcal{P}_N$ that involve unclean crashes in round N, in the sense that it (non-rigidly) maps such an "unclean" facet ϕ' to some face in the image $g(\phi)$ of some "clean" facet $\phi \in \mathcal{P}_N$, where the processes that crashed uncleanly in ϕ' crashed cleanly in ϕ or not at all.

In more detail, g maps a facet ϕ' to the maximal face ρ contained in the "full", i.e., unconstrained, image of ϕ' (that would be used without the discarding of "dirty" source vertices), where *no* vertex hears from a "dirty" witness of an unclean crash in ϕ'. Note that any such $\rho \neq \varnothing$ is also present as part of $g(\phi)$ for some facet ϕ where the crashing processes are not participating at all or are correct, so no "new" face needs to be included for mapping $g(\phi')$ here. On the other hand, there is no a priori guarantee that such a face ρ exists, as $g(\phi') = \varnothing$ is also possible; we will show in our non-emptyness proofs below (see Lemma 4.6 and Lemma 4.9) that this cannot happen under the conditions of our impossibility proof, however. In fact, this very intuitive property of g is what enables g to completely replace the complicated analysis of [5] by a simple connectivity argument based on its non-emptyness.

To formally define g, we need the following notation.

- For a facet $\phi = \{(p_i, \lambda_i^N) \mid i \in I_\phi\}$ of $\mathcal{P}_N$, where $I_\phi \subseteq [n]$ and $|I_\phi| = n - t$, let

$$\breve{\phi} = \{(p_i, \lambda_i^N) \mid i \in I_\phi \wedge \mathsf{names}(\lambda_i^N) \setminus \mathsf{names}(\phi) \neq \varnothing\}$$

  be the set of "dirty vertices" in ϕ. That is, $\breve{\phi} \subseteq \phi$ is the set of vertices in ϕ where the corresponding processes received a message from a process that has crashed uncleanly in round N.
- For a vertex $(p_i, \lambda_i^M) \in \mathcal{P}_M$, let $\mathrm{hist}(\lambda_i^M)$ be the set of all vertices (p_j, λ_j^N) contained in the heard-of history λ_i^M, i.e., the ones have been received by p_i directly or indirectly in any of the rounds $N+1, \ldots, M$.

▶ **Definition 4.1** (Agreement overhead carrier map)**.** *We define the carrier map g as follows:*

- *For a facet $\phi \in \mathcal{P}_N$,*

$$g(\phi) = \big\{\{(p_i, \lambda_i^M) \mid i \in I_\phi \wedge \mathrm{hist}(\lambda_i^M) \cap \breve{\phi} = \varnothing\}\big\}. \tag{5}$$

  That is, $g(\phi)$ is the (possibly empty) face of $\mathcal{P}_M$ consisting of the vertices that do not have any vertex in $\breve{\phi}$ in their heard of history.
- *For a face $\sigma \in \mathcal{P}_N$,*

$$g(\sigma) = \{\rho\} \text{ with } \rho = \text{maximal simplex in } \textstyle\bigcap_{\phi \in T_\sigma} g(\phi) \text{ s.th. } \mathsf{names}(\rho) \subseteq \mathsf{names}(\sigma), \tag{6}$$

  where T_σ denotes the set of all facets $\phi \in \mathcal{P}_N$ satisfying $\sigma \subseteq \phi$.

Note that Equation (5) and Equation (6) are consistent, in the sense that the image $g(\sigma)$ of a facet $\sigma \in \mathcal{P}_N$ is the same for both definitions, since $T_\sigma = \{\sigma\}$ here.

The facets of our source complex $\mathcal{P}_N$ need to satisfy the following additional conditions:

▶ **Definition 4.2.** *We define conditions* ***C1*** *and* ***C2*** *as follows:*
C1: *There is a facet $\phi \in \mathcal{P}_N$ with* $\mathsf{names}(\phi) = \Pi \setminus S$*, for every subset S of t processes.*
C2: *For every $\sigma \in \mathcal{P}_N$, $\sigma = \bigcap_{\phi \in T_\sigma} \phi$.*

Condition **C1** is satisfied for our source complexes, since both $\mathcal{P}_N = \mathrm{skel}_{n-t-1}\big(\Psi(n,k+1)\big)$ and $\mathcal{P}_N = \mathcal{K}_N$ contain every possible $(n-t-1)$-face by definition/construction.

The special context in which strictness of g is actually utilized, namely, [12, Lem. 13.4.2] (see Section 3.2 for details) is restricted to faces which are solely facet intersections, which actually makes **C2** in Definition 4.2 superfluous. However, since it is guaranteed for our source complexes, we can safely require it. Indeed, the regularity of the source complex $\mathcal{P}_N = \mathrm{skel}_{n-t-1}\big(\Psi(n,k+1)\big)$ trivially guarantees (C2), and for the source complex $\mathcal{P}_N = \mathcal{K}_N$, condition (C2) is easy to prove since every k-subset of the alive processes in $\mathcal{K}_{N-1}$ is crashed in round N in order to produce some facet $\phi \in \mathcal{P}_N$. So if $v = (p, \lambda_q) \in \bigcap_{\phi \in T_\sigma} \phi \setminus \sigma$ would exist, consider any facet $\phi \in T_\sigma$ where some process $q \notin \mathsf{names}(\phi)$ has crashed uncleanly after successfully sending his λ_q to everybody in round N in ϕ. There must also be a facet $\phi' \in \mathcal{P}_N$, which is the same as ϕ, except that $(q, \lambda_q) \in \phi'$ but $v \notin \phi'$ since p has crashed uncleanly after successfully sending λ_p to everybody in round N in ϕ'. Since $\sigma \subseteq \phi$ and hence $\phi' \in T_\sigma$ as well, we get the desired contradiction.

▷ Claim 4.3. g is a carrier map.

Proof. Let σ_1 and σ_2 be two faces of $\mathcal{P}_N$ with $\sigma_1 \subseteq \sigma_2$. We have $T_{\sigma_2} \subseteq T_{\sigma_1}$, and $\mathsf{names}(\sigma_1) \subseteq \mathsf{names}(\sigma_2)$, from which it follows that $g(\sigma_1) \subseteq g(\sigma_2)$. ◁

In Claim 4.4 below, we will prove that g is also strict. Our proof will rely on an essential property of g, namely, that if two facets ϕ_1 and ϕ_2 share a vertex $y = (p, \lambda_p) \in \phi_1 \cap \phi_2$, then the information p propagates to other common vertices in $\phi_1 \cap \phi_2$ in rounds $N+1, \ldots, M$ is the same in G_{ϕ_1} and G_{ϕ_2}. Recall that the graph G_ϕ is the product $G_{N+1,\phi} \circ G_{N+2,\phi} \circ \cdots \circ G_{M,\phi}$, where $G_{N+1,\phi}, \ldots, G_{M,\phi}$ are the graphs induced by $\mathsf{names}(\phi)$ in the sequence of graphs $G_{N+1}, G_{N+2}, \ldots, G_M$ that govern rounds $N+1, \ldots, M$. Whereas this is trivially satisfied when the source complex is $\mathrm{skel}_{n-t-1}\big(\Psi(n,k+1)\big)$, as there are no unclean crashes since all absent processes are initially dead, it needs to be secured by "discarding" vertices in $\breve{\phi}_1$ (resp., $\breve{\phi}_2$) in Equation (5) when unclean crashes may have happened. Indeed, p could have a predecessor q that sent its value λ_q to p in round N because $q \in \mathsf{names}(\phi_1)$ in ϕ_1, whereas it did not so in ϕ_2 because it crashed uncleanly, so that $q \notin \mathsf{names}(\phi_2)$.

▷ Claim 4.4. g is a strict carrier map.

Proof. Let ϕ_1, ϕ_2 be facets of $\mathcal{P}_N$, and let $\sigma = \phi_1 \cap \phi_2$. We show that $g(\phi_1) \cap g(\phi_2) = g(\sigma)$ by proving that $g(\phi_1) \cap g(\phi_2) \subseteq g(\sigma)$. Assume by contradiction that there exists a vertex $x \in g(\phi_1) \cap g(\phi_2)$, but $x \notin g(\sigma)$. Since $\sigma = \phi_1 \cap \phi_2$, and thanks to the definition of T_σ, $\sigma = \cap_{\phi \in T_\sigma} \phi$ must obviously hold. By the definition of g, we then get $g(\sigma) = \cap_{\phi \in T_\sigma} g(\phi)$. Hence, there must be some $\phi_3 \in T_\sigma$ such that $x \notin g(\phi_3)$, as well as a unique $x' \in \sigma$ with $\mathsf{names}(x') = \mathsf{names}(x)$.

Since $x \in g(\phi_1) \cap g(\phi_2)$, node $\mathsf{names}(x)$ can only hear from nodes in $\mathsf{names}(\sigma)$ in the subgraphs G_{ϕ_1} and G_{ϕ_2} of G induced by $\mathsf{names}(\phi_1)$ and $\mathsf{names}(\phi_2)$, respectively. On the other hand, $\mathsf{names}(x)$ must hear from a process outside $\mathsf{names}(\sigma)$ in G_{ϕ_3}. Indeed, assume for

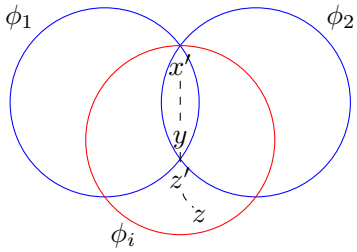


Figure 1 Illustration of the strictness proof in Claim 4.4.

a contradiction that this is not the case. Then, in any of the graphs $G_{\phi_1}, G_{\phi_2}, G_{\phi_3}$, node $\mathsf{names}(x)$ hears only from nodes in $\mathsf{names}(\sigma)$, and all nodes in $\mathsf{names}(\sigma)$ are of course included in $G_{\phi_1}, G_{\phi_2}, G_{\phi_3}$. Node $\mathsf{names}(x)$ hence has the same heard-of history in $G_{\phi_1}, G_{\phi_2}, G_{\phi_3}$, contradicting the assumption $x \notin g(\phi_3)$. Therefore, there must exist some $z \in \phi_3$ with $z \notin \phi_1$ and $z \notin \phi_2$ such that $\mathsf{names}(x)$ hears from $\mathsf{names}(z)$ in G_{ϕ_3} via some path P in round $N+1, \dots, M$ (see Figure 1 for an illustration). Let $y \in \sigma$ be such that

- $\mathsf{names}(y)$ is a node in P that is the closest to $\mathsf{names}(x') = \mathsf{names}(x)$ in G_{ϕ_3}, and
- $\mathsf{names}(y)$ has a neighbor $\mathsf{names}(z') \in P$ and $z' \notin \sigma$.

We must have $z' \notin \phi_1$. Indeed, if $z' \in \phi_1$, then the path suffix $P' \subseteq P$ leading from $\mathsf{names}(z') \to \mathsf{names}(y) \to \mathsf{names}(x)$ would be in G_{ϕ_1}, so $\mathsf{names}(x)$ would hear from $\mathsf{names}(z') \notin \mathsf{names}(\sigma)$ in G_{ϕ_1} in rounds $N+1, \dots, M$, which contradicts that it can only hear from nodes in $\mathsf{names}(\sigma)$ as established above. Analogously, $z' \notin \phi_2$ must hold. Note carefully, however, that the path suffix $P'' \subseteq P' \subseteq P$ leading from $\mathsf{names}(y) \to \mathsf{names}(x)$ is contained in any of $G_{\phi_1}, G_{\phi_2}, G_{\phi_3}$.

Since $z' \notin \phi_1 \cup \phi_2$ but $y \in \phi_1 \cap \phi_2 \cap \phi_3$, process $\mathsf{names}(z')$ must have crashed uncleanly in round N after sending to $\mathsf{names}(y)$ in both ϕ_1 and ϕ_2 (note that it is here where we need condition **C1** in Definition 4.2). However, in that case, Equation (5) would guarantee that $\mathsf{names}(x) \notin g(\phi_1) \cup g(\phi_2)$, which contradicts our initial assumption $x \in g(\phi_1) \cap g(\phi_2)$.

By monotonicity of g, it follows that $g(\phi_1) \cap g(\phi_2) = g(\phi_1 \cap \phi_2)$ for every two facets ϕ_1, ϕ_2 of $\mathcal{P}_N$.

We also need to prove strictness for faces, so let σ_1, σ_2 be two faces of $\mathcal{P}_N$ with $\sigma = \sigma_1 \cap \sigma_2$, and assume for a contradiction that there is some $x \in g(\sigma_1) \cap g(\sigma_2)$ but $x \notin g(\sigma)$. According to Equation (6), this implies that $x \in \phi_1 \cap \phi_2$ for any two facets $\phi_1 \in T_{\sigma_1}$ and $\phi_2 \in T_{\sigma_2}$, but that there is some facet $\phi_3 \in T_\sigma$ with $x \notin g(\phi_3)$. If we pick any such ϕ_1 and ϕ_2, we might observe $\phi' = \phi_1 \cap \phi_2 \supset \phi$, but still $x \in g(\phi_1) \cap g(\phi_2)$ but $x \notin g(\phi_3)$. It is easy to see, in particular, from Figure 1, that the above contradiction proof applies also here, since its arguments are not affected by assuming $\phi' \supset \phi$. ◁

We will now utilize our carrier map g for establishing our desired agreement overhead lower bound. As an appetizer, we will first provide a simple-to-prove eccentricity-based definition of a graph radius, which requires a static communication graph, i.e., $G = G_{N+1} = G_{G+2} = \dots = G_M$ in rounds $N+1, \dots, M$. Note that here, G_ϕ is equal to the $(M-N)$-th power of the subgraph of G induced by the processes present in ϕ.

▶ **Definition 4.5.** *For a node set $D \subseteq V$, the* eccentricity *of D in the graph $G = (V, E)$, denoted* $\mathrm{ecc}(D, G)$*, is the smallest integer d such that for every node $v \in V$ there is a path from some node $u \in D$ to v in G consisting of at most d hops.*
If G is a dynamic graph, $\mathrm{ecc}(D, G)$ is similarly defined, but with a temporal path from u to v; that is, if all nodes in D broadcast the same message in G by flooding, then all nodes in V receive the message in at most d rounds.

By this definition, if all nodes in D broadcast for $\mathrm{ecc}(D,G)$ rounds in the distributed message-passing model, every node in $V(G)$ hears from at least one node in D. By considering the set D that minimizes the eccentricity, we define a corresponding *Radius* $\mathrm{Rad}(G,t+k) = \min_{D\subseteq V,|D|=t+k} \mathrm{ecc}(D,G)$. This allows us to state the following essential property of the corresponding carrier map g:

▶ **Lemma 4.6.** *Let $R = M - N$. For a static communication graph G, if $R < \mathrm{Rad}(G,t+k)$, then $g : \mathcal{P}_N \to \mathcal{P}_M$ is a $(k-1)$-connected carrier map.*

▶ **Theorem 4.7.** *For every graph G, $t \geq 0$ and $k \geq 1$, there are no algorithms solving k-set agreement in the t-resilient model in G in less than $R = \lfloor \frac{t}{k} \rfloor + \mathrm{Rad}(G,t+k)$ rounds.*

Now we will finally turn to our ultimately desired lower bound, which essentially follows from the lower bound given in Theorem 4.7, by replacing $\mathrm{Rad}(G,t+k)$ with the (t,k)-radius $\mathrm{rad}(G,t,k)$ defined as follows:

▶ **Definition 4.8** ((t,k)-radius of a graph sequence G)**.** *For an n-node graph sequence $G = G_{N+1},\ldots,G_M$ and any two integers t,k with $t \geq 0$ and $k \geq 1$, we define the (t,k)-radius $\mathrm{rad}(G,t,k)$ as follows:*

$$\mathrm{rad}(G,t,k) = \min_{D,|D|=t+k} \max_{D'\subseteq D,|D'|=t} \mathrm{ecc}(D\setminus D', G\setminus D'). \tag{7}$$

Recall that $\mathrm{ecc}(D\setminus D', G\setminus D')$ is the number of rounds needed for $D\setminus D'$ to collectively broadcast in the subgraph sequence of G induced by $\Pi\setminus D'$.

Note carefully that this definition generalizes the definition of the (t,k)-radius of a static graph already given in Equation (1) to our graph sequences.

▶ **Lemma 4.9.** *Let $R = M-N$. If $R < \mathrm{rad}(G,t,k)$, then $g : \mathcal{P}_N \to \mathcal{P}_M$ is a $(k-1)$-connected carrier map.*

Exactly the same proof as for Theorem 4.7 thus yields the refined lower bound stated in the following theorem:

▶ **Theorem 4.10.** *For every graph G, $t \geq 0$ and $k \geq 1$, there are no algorithms solving k-set agreement in G in the t-resilient model in strictly less than $R = \lfloor \frac{t}{k} \rfloor + \mathrm{rad}(G,t,k)$ rounds.*

Our analysis also provides a lower bound for systems with t initially dead processes, by starting from the source complex $\mathcal{P}_N = \mathrm{skel}_{n-t-1}\big(\Psi(\{(p_i,[k+1]) \mid i\in[n]\})\big)$, which is shellable according to Theorem 3.6. Analogous to Theorem 4.10, this concludes in the following theorem.

▶ **Theorem 4.11.** *For every graph G, $t \geq 0$ and $k \geq 1$, there are no algorithms solving k-set agreement in G with t initially dead processes in less than $\mathrm{rad}(G,t,k)$ rounds.*

Note that we establish in [11] that Theorem 4.11 for $t = 0$ (almost) coincides with the dominance number based lower bound for k-set agreement in the KNOW-ALL model established in [5].

5 Upper Bound for Fixed Graphs

Let $G = (V,E)$ be an n-node graph with vertex connectivity $\kappa(G)$. Let $t < \kappa(G)$ be a non-negative integer, and let $k \geq 1$ be an integer. We are interested in solving k-set agreement in G with at most t crash failures. For $S \subseteq V$, let $G\setminus S$ denote the subgraph of G induced by

the nodes in $V \setminus S$, i.e., $G \setminus S$ is an abbreviation for $G[V \setminus S]$. For every graph H, let $D(H)$ denote its diameter. We define $D(G,t) = \max_{S \subseteq V, |S| \leq t} D(G \setminus S)$. Note that since $t < \kappa(G)$, and the maximization is over all sets S of size at most t, $D(G,t)$ is finite.

▶ **Theorem 5.1.** *There exists an algorithm solving k-set agreement in G in $\lfloor \frac{t}{k} \rfloor + D(G,t)$ rounds.*

Note that the bound in Theorem 5.1 matches the bound $\lfloor \frac{t}{k} \rfloor + 1$ rounds for k-set agreement in the n-node clique K_n under the synchronous t-resilient model (see [8]), as $D(K_n, t) = 1$.

Examples

- Let us consider the n-node cycle, i.e., $G = C_n$, with $t = 1$, and $k = 1$ (i.e., consensus). We have $D(C_n, 1) = n - 2$, as, for every node v, $C_n \setminus \{v\}$ is a path with $n - 1$ nodes. The algorithm of Theorem 5.1 must thus perform min-flooding for $1 + (n - 2) = n - 1$ rounds to solve consensus in C_n. Intuitively, this appears to be the best that can be achieved as the node with the smallest input value may crash at the first round, by sending its value to just one of its neighbors, and then $n - 2$ additional rounds will be needed for this value to reach all nodes.
- Let us consider the d-dimensional hypercube Q_d, $d \geq 1$, with $n = 2^d$ nodes. We have $\kappa(Q_d) = d$, and there are d internally-disjoint paths of length at most $d + 1$ between any two nodes, which implies that $D(Q_d, d - 1) = d + 1$. The algorithm of Theorem 5.1 must thus perform min-flooding for $\lfloor \frac{t}{k} \rfloor + (d + 1)$ rounds to solve k-set agreement in the t-resilient hypercube Q_d.

6 Conclusions

We provided novel lower bounds for k-set agreement in synchronous t-resilient systems connected by an arbitrary directed communication network. Our lower bound combines the $\lfloor t/k \rfloor$ lower bound (which we generalized to arbitrary communication graphs) obtained for rounds where exactly t processes crash with an additional novel lower bound on the agreement overhead caused by an arbitrary network, i.e., different from the complete graph. Our results use the machinery of combinatorial topology for studying the (high) connectivity properties of the round-by-round protocol complexes obtained by some novel and strikingly simple carrier maps, which we firmly believe to have applications also in other contexts. Whereas we also provided some upper bound result, the challenging question of possible tightness is deferred to future research.

References

1 Dan Alistarh, James Aspnes, Faith Ellen, Rati Gelashvili, and Leqi Zhu. Why extension-based proofs fail. *SIAM Journal on Computing*, 52(4):913–944, 2023. doi:10.1137/20M1375851.

2 Hagit Attiya, Armando Castañeda, and Sergio Rajsbaum. Locally solvable tasks and the limitations of valency arguments. *J. Parallel Distributed Comput.*, 176:28–40, 2023. doi:10.1016/J.JPDC.2023.02.002.

3 Hagit Attiya, Pierre Fraigniaud, Ami Paz, and Sergio Rajsbaum. On the existence of extension-based proofs of impossibility for set-agreement. In *32nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 15671 of *LNCS*, pages 56–73. Springer, 2025. doi:10.1007/978-3-031-91736-3_4.

4 Anders Björner and Michelle L. Wachs. Shellable nonpure complexes and posets. i. *Transactions of the American Mathematical Society*, 348(4):1299–1327, 1996. URL: http://www.jstor.org/stable/2155143.

5 Armando Castañeda, Pierre Fraigniaud, Ami Paz, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. A topological perspective on distributed network algorithms. *Theoretical Computer Science*, 849:121–137, 2021. doi:10.1016/J.TCS.2020.10.012.

6 Armando Castañeda, Pierre Fraigniaud, Ami Paz, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. Synchronous t-resilient consensus in arbitrary graphs. *Inf. Comput.*, 292:105035, 2023. doi:10.1016/J.IC.2023.105035.

7 Soma Chaudhuri. More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems. *Information and Control*, 105(1):132–158, July 1993. doi:10.1006/INCO.1993.1043.

8 Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. Tight bounds for k-set agreement. *J. ACM*, 47(5):912–943, 2000. doi:10.1145/355483.355489.

9 Pierre Fraigniaud, Minh Hang Nguyen, and Ami Paz. Agreement tasks in fault-prone synchronous networks of arbitrary structure. In *42nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 327 of *LIPIcs*, pages 34:1–34:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.STACS.2025.34.

10 Pierre Fraigniaud, Minh-Hang Nguyen, and Ami Paz. A simple lower bound for set agreement in dynamic networks. In *8th Symposium on Simplicity in Algorithms (SOSA)*, pages 253–262. SIAM, 2025. doi:10.1137/1.9781611978315.20.

11 Pierre Fraigniaud, Minh Hang Nguyen, Ami Paz, Ulrich Schmid, and Hugo Rincon Galeana. Lower bounds for k-set agreement in fault-prone networks, 2025. arXiv:2508.15562.

12 Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed computing through combinatorial topology*. Morgan Kaufmann, 2013.

A Proofs for Section 3 (Connectivity-Based Topological Analysis of Synchronous Systems)

▶ **Theorem 3.8.** *If $\mathcal{P}[J]$ is $(\dim(J)-1)$-connected for all $J \subseteq [k+1]$, then k-set agreement is not solvable with respect to $\mathcal{P}$.*

Proof. Let us assume, for the purpose of contradiction, that the k-set agreement task $\mathcal{T}$ is solvable with respect to $\mathcal{P}$. This implies there exists a simplicial map $\delta : \mathcal{P} \to \mathcal{O}$ that agrees with Δ. Let $\mathcal{K} = [k+1]$ viewed as a complex, and let $\Theta : \mathcal{K} \to 2^{\mathcal{P}}$ be defined as $\Theta(J) = \mathcal{P}[J]$ for every $J \subseteq [k+1]$ viewed as a simplex.

We claim that Θ is a carrier map. Indeed, if $J' \subseteq J$, then $\Psi(\{(p_i, J') \mid i \in [n]\}) \subseteq \Psi(\{(p_i, J) \mid i \in [n]\})$. That is, every $\sigma \in \Psi(\{(p_i, J') \mid i \in [n]\})$ belongs to $\Psi(\{(p_i, J) \mid i \in [n]\})$, which implies $\mathcal{P}[J'] \subseteq \mathcal{P}[J]$.

Thanks to Theorem 3.7.7(2) in [12], since Θ is a carrier map, and since, for each J, $\mathcal{P}[J]$ is $(\dim(J)-1)$-connected, we get that Θ has a simplicial approximation $(\mathrm{Div}(\mathcal{K}), g)$. That is:

- $\mathrm{Div}(\mathcal{K})$ is a chromatic subdivision of $\mathcal{K}$,
- $g : \mathrm{Div}(\mathcal{K}) \to \mathcal{P}$ is simplicial and chromatic, and
- for every $J \subseteq [k+1]$, and every $\rho \in \mathrm{Div}(J)$, $g(\rho) \in \Theta(J)$, where $\mathrm{Div}(J)$ is the subdivision of the face J of $\mathcal{K}$ induced by the global subdivision $\mathrm{Div}(\mathcal{K})$.

Let $f : \mathcal{P} \to \partial\mathcal{K}$ be defined as $f = \mathrm{val} \circ \delta$, where val is the trivial mapping that discards process IDs, and where $\partial\mathcal{K}$ is the boundary of $\mathcal{K}$. As the combination of two simplicial maps, we get that f is simplicial. Let $h : \mathrm{Div}(\mathcal{K}) \to \partial\mathcal{K}$ be defined as $h = f \circ g$.

We claim that h is a Sperner coloring of $\mathrm{Div}(\mathcal{K})$. For $J \subseteq [k+1]$ and $\rho \in \mathrm{Div}(J)$, $h(\rho) = f \circ g(\rho)$. Since $g(\rho) \in \Theta(J) = \mathcal{P}[J]$, there exists $\sigma \in \Psi(\{p_i, J) \mid i \in [n]\})$ such that $g(\rho) \in \mathcal{P}_\sigma$. The validity condition implies when processes in σ run alone, each output must be in $\mathrm{val}(\sigma) \subseteq J$, that is, for every $\tau \in \mathcal{P}_\sigma$, $\mathrm{val}(\delta(\tau)) \subseteq J$. Therefore $\mathrm{val} \circ \delta(g(\rho)) \subseteq J$.

Since there are no Sperner colorings of $\mathrm{Div}(\mathcal{K})$ that can avoid facets that are colored with $k+1$ colors, we get a contradiction to the assumption that $\mathcal{T}$ is solvable w.r.t. $\mathcal{P}$. ◀

▶ **Lemma 3.12.** *Let $q \geq 0$ be an integer, and let us consider a sequence of pure complexes, and carrier maps $\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \mathcal{K}_2$, where f_0 is a q-shellable carrier map, f_1 is a q-connected carrier map, and, for every $\sigma \in \mathcal{K}_0$ with $\mathrm{codim}(\sigma) \leq q+1$, $\mathrm{codim}(\sigma) \geq \mathrm{codim}(f_0(\sigma))$. Then, $f_1 \circ f_0$ is a q-connected carrier map.*

Proof. First, $g = f_1 \circ f_0$ is a strict carrier map because it is a composition of two strict carrier maps. It remains to check that $g(\sigma)$ is $(q - \mathrm{codim}\,\sigma)$-connected. Considering an arbitrary $\sigma \in \mathcal{K}_0$ satisfying $\mathrm{codim}(\sigma) \leq q+1$, we have:

- **(i)** $f_0(\sigma)$ is shellable and pure,
- **(ii)** $\mathrm{codim}(f_0(\sigma)) \leq \mathrm{codim}(\sigma)$,
- **(iii)** for every simplex $\tau \in f_0(\sigma)$, the co-dimension of τ in $f_0(\sigma)$, denoted by $\mathrm{codim}(\tau, f_0(\sigma))$, satisfies $\mathrm{codim}(\tau, f_0(\sigma)) = \dim(f_0(\sigma)) - \dim(\tau)$, and
- **(iv)** $\mathrm{codim}(\tau, \mathcal{K}_1) = \mathrm{codim}(\tau, f_0(\sigma)) + \mathrm{codim}(f_0(\sigma), \mathcal{K}_1)$.

Let $q' = q - \mathrm{codim}(f_0(\sigma))$. Since f_1 is a q-connected carrier map, $f_1(\tau)$ is $(q - \mathrm{codim}(\tau, \mathcal{K}_1))$-connected. Equivalently, $f_1(\tau)$ is $(q' - \mathrm{codim}(\tau, f_0(\sigma)))$-connected. By applying Lemma 3.10, $f_1(f_0(\sigma))$ is q' connected. Thus, $g(\sigma)$ is $(q - \mathrm{codim}\,\sigma)$-connected since $q' \leq q - \mathrm{codim}(\sigma)$ by (ii). ◀

▶ **Lemma 3.13.** *Let $q \geq 0$ and $\ell \geq 0$ be integers, and let us consider a sequence of pure complexes, and carrier maps $\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \ldots \xrightarrow{f_\ell} \mathcal{K}_{\ell+1}$, such that the carrier maps $f_0, \ldots, f_{\ell-1}$ are q-shellable, the carrier map f_ℓ is q-connected, and, for every $i \in \{0, \ldots, \ell-1\}$, and every $\sigma \in \mathcal{K}_i$ with $\mathrm{codim}(\sigma) \leq q+1$, $\mathrm{codim}(\sigma) \geq \mathrm{codim}(f_i(\sigma))$. Then, $f_\ell \circ \cdots \circ f_0$ is a q-connected carrier map.*

Proof. We use induction on $k \geq 0$ to prove that $g_k = f_\ell \circ \cdots \circ f_{\ell-k}$ is a q-connected carrier map. Note that $g_\ell = f_\ell \circ \cdots \circ f_0$. For the base case $k = 0$, the claim is immediate from our assumption on f_ℓ. For the induction step from k to $k+1$, we note that $g_{k+1} = f_\ell \circ \cdots \circ f_{\ell-k-1} = g_k \circ f_{\ell-k-1}$. Since $f_{\ell-k-1}$ is q-shellable and guarantees $\mathrm{codim}(\sigma) \geq \mathrm{codim}(f_{\ell-k-1}(\sigma))$ for all $\sigma \in \mathcal{K}_{\ell-k}$ with $\mathrm{codim}(\sigma) \leq q+1$ by our assumptions, and since g_k is q-connected by the induction hypothesis, we can apply Lemma 3.12, which ensures that $g_{k+1} = g_k \circ f_{\ell-k-1}$ is q-connected as needed. ◀

▶ **Lemma 3.14.** *For every $i \in \{0, \ldots, N-1\}$, f_i is $(k-1)$-shellable, and, for every $\sigma \in \mathcal{K}_i$ with $\mathrm{codim}(\sigma) \leq k$, $\mathrm{codim}(f_i(\sigma)) \leq \mathrm{codim}(\sigma)$.*

Proof. Lemma 13.5.5 in [12] shows that one can define a shelling order on the facets of $f_i(\sigma)$, which is a pure complex by Equation (4), for every $\sigma \in \mathcal{K}_i$ that yields $f_i(\sigma) \neq \varnothing$. All that remains to be proved is hence strictness, and the additional condition $\mathrm{codim}\, f_i(\sigma) \leq \mathrm{codim}\,\sigma$ for every $\sigma \in \mathcal{K}_i$ with $\mathrm{codim}\,\sigma \leq k$.

For the latter, note that, for every $\sigma \in \mathcal{K}_i$, if $\dim(\sigma) < n - k(i+1)$, then $f_i(\sigma) = \varnothing$. Since Equation (4) implies that $\dim(\mathcal{K}_{i+1}) = \dim(\mathcal{K}_i) - k$, we can indeed guarantee $\mathrm{codim}(f_i(\sigma)) \leq \mathrm{codim}(\sigma)$ for every simplex σ in $\mathcal{K}_i$ satisfying $\mathrm{codim}(\sigma) \leq k$.

For strictness, let ϕ_1, ϕ_2 be simplices of $\mathcal{K}_i$, and let $\phi = \phi_1 \cap \phi_2$. We prove that $f_i(\phi) = f_i(\phi_1) \cap f_i(\phi_2)$. We have

$$f_i(\phi_1) = \bigcup_{\tau \in \mathrm{Face}_{n-1-k(i+1)}\, \phi_1} \Psi(\tau, [\tau, \phi_1]), \quad \text{and} \quad f_i(\phi_2) = \bigcup_{\tau \in \mathrm{Face}_{n-1-k(i+1)}\, \phi_2} \Psi(\tau, [\tau, \phi_2]).$$

If $f_i(\phi_1) \cap f_i(\phi_2) = \varnothing$, then $f_i(\phi) = \varnothing$ by the monotonicity of carrier map f_i. So let us assume that $f_i(\phi_1) \cap f_i(\phi_2) \neq \varnothing$. Let us then consider an arbitrary simplex $\sigma \in f_i(\phi_1) \cap f_i(\phi_2)$. There exists $\tau \in \mathrm{Face}_{n-1-k(i+1)}\, \phi_1$ and $\tau' \in \mathrm{Face}_{n-1-k(i+1)}\, \phi_2$ such that

$$\sigma \in \Psi(\tau, [\tau, \phi_1]) \cap \Psi(\tau', [\tau', \phi_2]).$$

This implies that there exists a simplex $\tau'' \subseteq \tau \cap \tau' \subseteq \phi_1 \cap \phi_2$ such that

$$\sigma \in \Psi(\tau'', [\tau, \phi_1]) \cap \Psi(\tau'', [\tau', \phi_2]) \subseteq \Psi(\tau'', [\tau, \phi_1] \cap [\tau', \phi_2]).$$

We claim that τ and τ' are faces of ϕ. Indeed, if τ (which is a face of ϕ_1) is not a face of ϕ, then τ is not a face of ϕ_2. Then, $[\tau, \phi_1] \cap [\tau', \phi_2] = \varnothing$, which contradicts the fact that $f_i(\phi_1) \cap f_i(\phi_2) \neq \varnothing$. Consequently,

$$\begin{aligned}
\sigma \in \Psi(\tau'', [\tau, \phi_1] \cap [\tau', \phi_2]) &\subseteq \Psi(\tau'', [\tau, \phi] \cap [\tau', \phi]) \\
&= \Psi(\tau'', [\tau \cup \tau', \phi]) && (8) \\
&\subseteq \bigcup_{\rho \in \mathrm{Face}_{n-1-k(i+1)}\, \phi} \Psi(\tau'', [\rho, \phi]) \\
&\subseteq \bigcup_{\rho \in \mathrm{Face}_{n-1-k(i+1)}\, \phi} \Psi(\rho, [\rho, \phi]) \\
&= f_i(\phi). && (9)
\end{aligned}$$

where Equation (8) follows from the fact that τ (resp., τ') is a face of every simplex in $[\tau, \phi]$ (resp., $[\tau', \phi]$). Equation (9) implies that $f_i(\phi_1) \cap f_i(\phi_2) \subseteq f_i(\phi)$, from which $f_i(\phi_1) \cap f_i(\phi_2) = f_i(\phi)$ follows by monotonicity of carrier maps. ◀

▶ **Theorem 3.16.** *For integers $t \geq 0$ and $k \geq 1$, let $\mathcal{P}$ be the protocol complex of k-set agreement after $N = \lfloor \frac{t}{k} \rfloor$ rounds in the synchronous t-resilient model with the complete communication graph, where exactly k processes crash per round. For every $J \subseteq [k+1]$, $\mathcal{P}[J]$ is a $(\dim(J) - 1)$-connected subcomplex.*

Proof. Let $J \subseteq [k+1]$. For every $i = 1, \ldots, \lfloor \frac{t}{k} \rfloor$, let $\mathcal{P}^{(i)}[J]$ be the protocol complex after i rounds, starting from $\mathcal{P}^{(0)}[J] = \mathcal{I}[J] = \Psi(\{(p_i, J) \mid i \in [n]\})$ and $\mathcal{P}[J] = \mathcal{P}^{(N)}[J]$. By construction, $\mathcal{K}_0 = \mathcal{P}^{(0)}[J] = \mathcal{I}[J], \mathcal{K}_1 = \mathcal{P}^{(1)}[J], \ldots, \mathcal{K}_N = \mathcal{P}^{(N)}[J]$ are the complexes induced by the carrier map $f_i : \mathcal{K}_i \to \mathcal{K}_{i+1}$ given by Equation (4), which crashes exactly k processes in round $i+1$, $0 \leq i < N$. Consider the sequence

$$\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{N-1}} \mathcal{K}_N \xrightarrow{id} \mathcal{K}_N. \tag{10}$$

By Lemma 3.15, we have $f_{N-1} \circ \cdots \circ f_0 : \mathcal{K}_0 \to \mathcal{K}_N$ is a $(k-1)$-connected carrier map. Since the input complex $\mathcal{I}[J]$ is a pseudosphere, and hence pure and shellable, Lemma 3.10 implies that $\mathcal{K}_N = \mathcal{P}[J]$ is $(k-1)$-connected. Therefore, $\mathcal{P}[J]$ is also $(\dim(J) - 1)$-connected. ◀

B Proofs for Section 4 (A Lower Bound for the Agreement Overhead for Arbitrary Graphs)

▶ **Lemma 4.6.** *Let $R = M - N$. For a static communication graph G, if $R < \mathrm{Rad}(G, t+k)$, then $g : \mathcal{P}_N \to \mathcal{P}_M$ is a $(k-1)$-connected carrier map.*

Proof. We show that for every face σ of $\mathcal{P}_N$ with $codim(\sigma) \leq k$, $g(\sigma) \neq \varnothing$. Then, since $g(\sigma)$ is a face of $\mathcal{P}_M$, $g(\sigma)$ is $(k-1)$-connected. Recall that $g(\sigma) = \{\rho\}$, where ρ is the maximal simplex in $\bigcap_{\phi \in T_\sigma} g(\phi)$ satisfying $\mathsf{names}(\rho) \subseteq \mathsf{names}(\sigma)$.

Let $S = V(G) \setminus \mathsf{names}(\sigma)$, $|S| \leq t + k$. Since $R < \mathrm{Rad}(G, t+k)$, there is a node $p \in \mathsf{names}(\sigma)$ such that p does not hear from any node in S after R rounds in G. It implies that, for every $\phi \in T_\sigma$, node p does not hear from S in G_ϕ in rounds $N+1, \ldots, M$. There is hence a vertex x with $\mathsf{names}(x) = p \in \mathsf{names}(\sigma)$ and $x \in \bigcap_{\phi \in T_\sigma} g(\phi)$. Thus, $x \in g(\sigma) \neq \varnothing$ as claimed. ◀

▶ **Theorem 4.7.** *For every graph G, $t \geq 0$ and $k \geq 1$, there are no algorithms solving k-set agreement in the t-resilient model in G in less than $R = \lfloor \frac{t}{k} \rfloor + \mathrm{Rad}(G, t+k)$ rounds.*

Proof. The proof is literally the same as the one for Theorem 3.16, except that it replaces Equation (10) by the following chain of carrier maps, with $N = \lfloor \frac{t}{k} \rfloor$ and $M = N + R$:

$$\mathcal{K}_0 \xrightarrow{f_0} \mathcal{K}_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{N-1}} \mathcal{K}_N = \mathcal{P}_N \xrightarrow{g} \mathcal{P}_M \tag{11}$$

Herein, $\mathcal{K}_0 = \mathcal{P}^{(0)}[J] = \mathcal{I}[J] = \Psi(P_i, J \mid i \in \{0, \ldots, n\})$ is again the input complex, $\mathcal{K}_1 = \mathcal{P}^{(1)}[J], \ldots, \mathcal{K}_N = \mathcal{P}^{(N)}[J]$ are the protocol complexes resulting from the N crashing rounds, $\mathcal{P}_N = \mathcal{K}_N$ is the source complex for our carrier map g, and finally $\mathcal{P}[J] = \mathcal{P}_M$ is the protocol complex reached from $\mathcal{P}_N$ after $M - N = \mathrm{Rad}(G, t+k)$ rounds. ◀

▶ **Lemma 4.9.** *Let $R = M - N$. If $R < \mathrm{rad}(G, t, k)$, then $g : \mathcal{P}_N \to \mathcal{P}_M$ is a $(k-1)$-connected carrier map.*

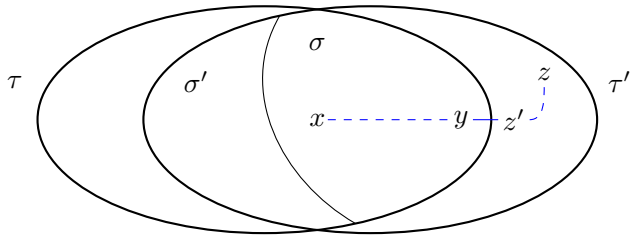


Figure 2 Two facets τ, τ' in $\mathcal{P}_N$ with $\tau \cap \tau' = \sigma'$. The vertices x resp. y resp. $\{z', z\}$ belong to σ resp. $\sigma \subseteq \sigma'$ resp. τ' as shown.

Proof. Let σ be a face of $\mathcal{P}_N$ with $codim(\sigma) \leq k$. It suffices to show that $g(\sigma) \neq \varnothing$: since $g(\sigma)$ is a face of $\mathcal{P}_M$, $g(\sigma)$ must be $(k-1)$-connected.

For every facet τ of $\mathcal{P}_N$, recall that G_τ denotes the subgraph sequence of G induced by $\mathsf{names}(\tau)$. Choose $D = \Pi \setminus \mathsf{names}(\sigma)$, which must satisfy $t \leq |D| \leq t + k$. Due to condition **C1** in Definition 4.2, there is indeed a facet τ containing σ in $\mathcal{P}_N$ such that $\mathsf{names}(\tau) = \Pi \setminus D'$. Since $R < \mathrm{rad}(G, t, k)$, there is hence a process $p \in \mathsf{names}(\sigma)$, and $D' \subseteq D, |D'| = t$ such that p does not hear from any process in $D \setminus D'$ in G_τ in rounds $N+1, \ldots, M$.

Let $x = (p, \lambda_p) \in \sigma$ be the vertex corresponding to p. So even if every node broadcasts in G_τ during rounds $N+1, \ldots, M$, node $p \in \mathsf{names}(\sigma)$ does not hear from any process in $\mathsf{names}(\tau) \setminus \mathsf{names}(\sigma)$.

Now assume that there is a facet $\tau' \supseteq \sigma$ in $\mathcal{P}_N$ such that process $p = \mathsf{names}(x)$ hears from a process $\mathsf{names}(z) \in \mathsf{names}(\tau') \setminus \mathsf{names}(\sigma)$ in $G_{\tau'}$ in rounds $N+1, \ldots, M$, see Figure 2 for an illustration. Define $\sigma' = \tau \cap \tau' \supseteq \sigma$. Let P be a path in $G_{\tau'}$ (of course of length less or equal to R) leading from $\mathsf{names}(z) \to \mathsf{names}(x)$, and let $\mathsf{names}(z') \in P \setminus \mathsf{names}(\sigma)$

be the node closest to $\mathsf{names}(x)$ in P outside σ. If $\mathsf{names}(z')$ belonged to $\mathsf{names}(\sigma')$, then $\mathsf{names}(x)$ would hear from $\mathsf{names}(z')$ also in G_τ within R rounds, through the path suffix $P' \subseteq P$ going from $\mathsf{names}(z') \to \mathsf{names}(x)$, which contradicts our assumption. Thus, $\mathsf{names}(z') \in \mathsf{names}(\tau') \setminus \mathsf{names}(\tau)$. Consequently, the path P' from $\mathsf{names}(z')$ to $\mathsf{names}(x)$ only contains $\mathsf{names}(z')$ and processes from $\mathsf{names}(\sigma)$. Let $\mathsf{names}(y) \in P \cap \mathsf{names}(\sigma)$ be the neighbor of z' in P contained in σ, i.e., $\mathsf{names}(z') \in \mathrm{In}_{\mathsf{names}(y)}(G_N)$ in round N: Indeed, in the scenario corresponding to the facet τ, $\mathsf{names}(z')$ is dead in $\mathcal{P}_N$, but alive in the scenario corresponding to τ'. Therefore, in round N, process $\mathsf{names}(y)$ hears from $\mathsf{names}(z')$ in $G_{\tau'}$, but does not hear from $\mathsf{names}(z')$ in G_τ. But then, according to g's "discarding" of "dirty" source vertices in Equation (5), $y \notin \sigma'$, contradicting our assumption.

Thus, for every facet $\tau' \supseteq \sigma$ in $\mathcal{P}_N$, $\mathsf{names}(x)$ does not hear from any process in $\mathsf{names}(\tau') \setminus \mathsf{names}(\sigma)$ in $G_{\tau'}$ in rounds $N+1, \ldots, M$. Consequently, there must be a vertex x in σ with $\mathsf{names}(x) = p$, and $x \in g(\sigma)$. So, $g(\sigma) \neq \varnothing$. ◀

C Proofs for Section 5 (Upper Bound for Fixed Graphs)

▶ **Theorem 5.1.** *There exists an algorithm solving k-set agreement in G in $\lfloor \frac{t}{k} \rfloor + D(G,t)$ rounds.*

Proof. The algorithm and its proof of correctness are directly inspired from the k-set agreement algorithm for the clique K_n in [8], and from its analysis. The algorithm is merely the min-flooding algorithm for $\lfloor \frac{t}{k} \rfloor + D(G,t)$ rounds. That is, every node sends its input value to all its neighbors at the first round, and, at each round $r \geq 2$, every node forwards the minimum value received so far to all its neighbors. After $\lfloor \frac{t}{k} \rfloor + D(G,t)$ rounds, every node outputs the smallest value it became aware of during the whole execution of the protocol, which may be its own input value, or the input value of another node received during min-flooding.

Termination and validity are satisfied by construction. We now show that at most k values are outputted in total by the (correct) nodes. Let $r \in \{1, \ldots, \lfloor \frac{t}{k} \rfloor\}$, and let us consider the system after $r-1$ rounds of min-flooding have been performed. We focus on the nodes that have not crashed during the first $r-1$ rounds, and, among these nodes, we consider those that are holding the smallest values currently in the system. More precisely, let $U \subseteq V$ be a set of k nodes that have not crashed during the first $r-1$ rounds, and satisfying that, for every value x held by a node $u \notin U$ that has not crashed during the first $r-1$ rounds, x is at least as large as any value currently hold by the nodes in U.

We claim that, if some node $u \in U$ does not crash at round r and holds a value x then, then by the end of round $r + D(G,t)$ each correct node will either know x or a smaller value. Indeed, if u does not crash at round r, then, at this round, u sends x to all its (correct) neighbors. Since $t < \kappa(G) \leq \deg(u)$, we have that, for every suffix of the current execution, at least one neighbor u' of u is correct, i.e., one correct node u' holds x at the end of round r. It follows that all the correct nodes will have received x or smaller values by the end of round $r + D(G,t)$.

As a consequence of the claim, if less than k nodes crash at some round $r \in \{1, \ldots, \lfloor \frac{t}{k} \rfloor\}$, then at the end of round $r + D(G,t)$, every correct node knows at least one value among the smallest k values present in the system at the end of round $r-1$. This guarantees that at most k distinct values are outputted by the nodes.

On the other hand, for all executions in which at least k nodes crash in each of the first $\lfloor \frac{t}{k} \rfloor$ rounds, less that k nodes can crash at round $\lfloor \frac{t}{k} \rfloor + 1$. So, let v be a node that does not crash at round $\lfloor \frac{t}{k} \rfloor + 1$, and that holds one of the smallest k values in the system after

$\lfloor \frac{t}{k} \rfloor$ rounds, say x. Round $\lfloor \frac{t}{k} \rfloor + 1$ can be viewed as the first round of broadcast of value x from node v. This broadcast will complete in $D(G,t)$ rounds in total, no matter which nodes distinct from v crashes at rounds $r \geq \lfloor \frac{t}{k} \rfloor + 1$, and no matter whether v itself crashes at some round $r > \lfloor \frac{t}{k} \rfloor + 1$. Therefore, at the end of round $\lfloor \frac{t}{k} \rfloor + D(G,t)$, all correct nodes have received at least one value among the smallest k values present in the system at the end of round $\lfloor \frac{t}{k} \rfloor$. This guarantees that at most k distinct values are outputted by the nodes. ◀

Coordination Through Stochastic Channels

Pierre Fraigniaud
Inst. de Recherche en Informatique Fondamentale (IRIF), CNRS and Université Paris Cité, France

Boaz Patt-Shamir
School of Electrical Engineering, Tel Aviv University, Israel

Sergio Rajsbaum
Instituto de Matemáticas, UNAM, Mexico City, Mexico

Abstract

We consider a *stochastic network* model consisting of a set of n synchronous processes communicating by message passing. In each round, processes send messages directly to each other over a complete communication graph. The processes do not fail, but messages can be lost. Each message is delivered with probability p, for a given parameter $p \in [0, 1]$. We study the following optimization version of approximate agreement in this model. We assume that processes start with binary input values, execute an algorithm for a fixed number of rounds, and decide values in $[0, 1]$ satisfying the usual *validity* requirement stating that if all processes start with the same input value, then they should all decide that value. We propose deterministic algorithms that minimize the expected *discrepancy*, namely, the expected maximum distance between the decided values. We also present lower bounds on the expected discrepancy, which demonstrate the optimality of our algorithms for two processes. Finally, we present applications of our algorithms to solve randomized consensus and randomized approximate agreement.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Distributed computing models; Theory of computation → Random network models; Theory of computation → Distributed algorithms

Keywords and phrases Approximate agreement, randomized consensus, stochastic models, topology

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.32

Funding *Pierre Fraigniaud*: Additional support from ANR projects DUCAT (ANR-20-CE48-0006), ENEDISC (ANR-24-CE48-7768-01), and PREDICTIONS (ANR-23-CE48-0010), and from the InIDEX Project METALG.
Boaz Patt-Shamir: This research was supported by the Israel Science Foundation, grant No. 1948/21.
Sergio Rajsbaum: Part of this work was done while visiting IRIF, France, supported by ENEDISC (ANR-24-CE48-7768-01) and Ecole Polytechnique Gaspard Monge.

1 Introduction

We consider a set of n synchronous processes communicating by sending messages to each other along the edges of a fully connected graph, i.e., for every two processes i and j, there is a directed channel from i to j, and a directed channel from j to i. The processes do not fail, but the channels may fail to deliver messages. In each round, for every directed channel, the message sent through that channel is delivered with probability p, for a given parameter $p \in [0, 1]$.

This model has been called in the past *stochastic (dynamic) network*, and also sometimes referred to as evolving or temporal stochastic graph model as, in each round, the set of channels through which the messages have been delivered may define a different directed graph. Such a model has been studied since the 1980s, including variants where the probability of the directed graph at each round my depend on the graphs that occurred during the previous rounds, where the random graph at each round may be undirected, or where a


© Pierre Fraigniaud, Boaz Patt-Shamir, and Sergio Rajsbaum;
licensed under Creative Commons License CC-BY 4.0



random graph distribution is assumed that guarantees that each instantaneous graph, or the union of a few consecutive graphs, is connected with high probability. We refer to the surveys and recent papers [5, 7, 11, 14, 16] for details and references. The stochastic network model is of theoretical interest (e.g., the directed Erdős-Réni random graph model can be viewed as a specific instance of the model), and also of practical interest, e.g. it has been argued that assuming that every link delivers a message independently with some probability p in every round is actually a quite realistic model for uncorrelated transient channel or network interface failures in homogeneous system architectures, e.g., wireless and ad hoc networks [17, 20, 21].

Various problems have been considered before in a stochastic dynamic network, mainly related to broadcast, gossiping, connectivity and routing, as described in the above cited surveys. In this paper we consider the following problem, which can be viewed as an optimization variant of the classic approximate agreement problem [6]. We assume that processes start with binary input values, execute an algorithm for a fixed number of rounds, and decide values in $[0, 1]$ satisfying the standard *validity* requirement stating that (1) if all processes start with the same input value, then they should all decide that value, and (2) otherwise, each process can decide any value in $[0, 1]$. The *discrepancy* of the algorithm in an execution is the largest difference between any two decided values. Since executions are stochastic, the discrepancy is a random variable, and we seek algorithms with smallest *expected discrepancy* over all input assignments. For instance, in the case of two processes, and letting $q = p - 1$, the directed graph induced by the correct links (i.e., the channels through which the messages were delivered) is picked from $\mathcal{G} = \{\leftarrow\!\bullet, \rightarrow\!\bullet, \leftrightarrow\!\bullet, \circ\!\!-\!\!\bullet\}$, where the first two graphs are selected with probability pq each, the third with probability p^2 and the fourth with probability q^2.

1.1 Our Results

We describe algorithms minimizing the expected discrepancy, showing several interesting, surprising behaviors.

- We first study the case of two processes in detail, proving tight upper and lower bounds on the expected discrepancy. In particular, we prove a lemma, referred to as the Integrality Lemma, stating that, for optimal algorithms, it is sufficient to assume that the processes always decide integral values, either 0 or 1. We present two algorithms, one that is optimal for $p \leq 1/2$, and one that is optimal for $p \geq 1/2$. Our lower bound technique shows that stochastic settings can be analyzed by following the algebraic topology approach although the latter was initially designed for analyzing protocols in non-stochastic settings [12].
- When one considers executions conditioned on the event that at least one message is delivered, it is intuitively clear that, when p is large, i.e., when the probability of message delivery is high, it should be possible to obtain small expected discrepancy, namely, going down to 0 as p gets closer to 1. We confirm this intuition with our algorithms. However, we additionally show that this is also the case when $q = 1 - p$ is large, i.e. when the probability of message loss is high. As q gets closer to 1, the expected discrepancy of our algorithms also goes down to 0. In particular, when we discuss the application to randomized consensus, we will see that the probability of error goes down to 0, both when the probability of message loss is either very low or very high (conditioned on the event that at least one message is delivered).
- We then move on to the case of $n > 2$ processes. We show that the Integrality Lemma does not hold anymore, even for three processes. That is, an optimal algorithm must decide fractional values. As for the case of two processes, we show that there are thresholds for

the values of p such that, depending on whether the value of p is smaller or larger than each threshold, a different algorithm minimizes the expected discrepancy. As mentioned above, for two processes, there is a unique threshold $\theta = 1/2$, but we show that, for three processes, there are two thresholds $\theta_1 \approx 0.35$ and $\theta_2 = 1/2$. We provide optimal algorithms for three processes, one for each interval $[0, \theta_1]$, $[\theta_1, \theta_2]$, and $[\theta_2, 1]$ of message delivery probability p.

- For the case of a large number of processes n, we design a 1-round algorithm which guarantees an expected discrepancy that goes to 0 when n grows to infinity, except for highly unbalanced input configurations, i.e., when the number of 0s is $\omega(\log n)$, or when the number of 1s is $\omega(\log n)$. We however show how to adapt our 1-round algorithm for handling all possible input configurations, at the mere expense of one additional round.
- Finally, to support our claim that minimizing the expected discrepancy could be useful in applications where processes must decide values within a small ϵ from each other in most executions, and tolerate few executions with large discrepancy – as for clock synchronization, sensor replication, among others (see, e.g. [9]) – we provide some concrete applications. Specifically, we design algorithms solving randomized binary consensus, and randomized approximate agreement, with small error probability. In fact, for two processes, our randomized approximate agreement algorithm also applies to the case of arbitrary inputs in $[0, 1]$, or even in $\mathbb{R}$, and we can also prove corresponding lower bounds.

Alon et al. [1] consider a model similar to ours, from the information theoretic perspective. They assume binary symmetric channels (BSC), i.e each message is a bit that may be flipped with some constant probability $\varepsilon > 0$. Their concern is computing a function (while ours is computing a task). Roughly, they show that for $n \to \infty$, any computation over failure-free channels can be emulated, with high probability, over BSC channels with a constant multiplicative overhead. These results are asymptotic and are not applicable to small n values, but they inspired our Theorem 15.

Organization

The model is presented in Section 2. Algorithms for the case of two processes are analyzed in Section 3, and corresponding lower bounds in Section 4, where the integrality lemma is proved. In Section 5 we present our results for the case of more than 2 processes. We discuss applications in Section 6. The conclusions are in Section 7. Some proofs and extensions (i.e. message delivery probabilities that may vary between different channels, and in different rounds, and variance analysis) are omitted from this extended abstract.

2 Model

The *stochastic dynamic network* model involves a set of $n \geq 1$ processes labeled from 1 to n. For each ordered pair $(i, j) \in [n] \times [n]$ of distinct integers, there is a directed channel from process i to process j. Communication proceeds as a sequence of synchronous rounds. At each round $r \geq 1$, each of the n processes can send one message of arbitrary size to each of the other processes. Every message may however fail to reach its destination: it only succeeds with some probability. Specifically, for each round $r \geq 1$, and for every message sent by process i to process j,

$$\Pr[\text{the message sent by } i \text{ at round } r \text{ is received by } j] = p^r_{i,j} \ ,$$

where, for every $i, j \in [n]$ with $i \neq j$, and for every $r \geq 1$, $p^r_{i,j} \in [0, 1]$. When a message is not received, which occurs with probability $q^r_{i,j} = 1 - p^r_{i,j}$, it is lost. The sender of a message

is not informed of whether the message reached its destination or not. The probabilities $p^r_{i,j}$, for all $(i,j) \in [n] \times [n]$ and $r \geq 1$, are parameters of the model, and the n processes are aware of their values. We concentrate on the *uniform* case where there exists $p \in [0,1]$ such that, for every $(i,j) \in [n] \times [n]$, and every $r \geq 1$, $p^r_{i,j} = p$, and discuss how to generalize our results in the full version. When there are just two processes, they are referred to as *players*, and called Alice and Bob.

All our algorithms are deterministic. Note however that the outputs of a deterministic algorithm in our stochastic model are random due to the probabilistic nature of the message delivery. We consider *one-shot algorithms*, i.e., algorithms that execute a fixed number of rounds, and then each process outputs a value.

We study the following optimization version of approximate agreement, that we refer to as the *Agreement Optimization* problem.

▶ **Definition 1** (Agreement Optimization). *Each process $i \in \{1, \ldots, n\}$ starts with a private input $x_i \in \{0,1\}$, and decides a value $y_i \in [0,1]$ subject to the following two conditions:*
Termination: *Every process must terminate in a finite number of rounds, k.*
Validity: *If all processes start with the same input value x then all processes must decide x.*
The objective is to minimize discrepancy *defined as* $\max_{1 \leq i < j \leq n} |y_i - y_j|$.

In stochastic environments, the discrepancy is a random variable. We will concentrate on minimizing the *expected* discrepancy. In Section 6 we discuss how this implies minimizing probability of error in consensus and approximate agreement.

We will use several times the following consequence of the Validity condition.

▶ Remark 2. If processes have the same input value, then they all output that value in any execution. Furthermore, if some processes with input value x receives the same input value x from some set of processes, and nothing from other processes, then it has to decide x.

3 Two player algorithms

We assume in this section, without loss of generality, that Alice has input 0 and Bob has input 1. This assumption is solely for the purpose of presenting the algorithms; none of the two players initially know the input of the other player. We consider only initial configurations with different input values due to Remark 2.

We first consider single round algorithms in Section 3.1, and then extend our results to the case of multiple rounds, in Section 3.2.

3.1 Single round algorithms

In a single round, Alice and Bob send one message each, and hence there are four possible executions, according to whether each message was delivered or not. Remark 2 states that a player who did not receive a message must output its input value. So a single-round protocol can be completely characterized by the outputs made by the players as a response to the reception of a message containing a value different than their input (in all other cases, the output must be equal to the input).

Let us denote by $y_A \in [0,1]$ the output of Alice (with input 0) after receiving a message with value 1 from Bob, and similarly $y_B \in [0,1]$ denote the output of Bob (with input 1) after receiving value 0 from Alice. Each message is delivered with probability p, and dropped with probability $q = 1 - p$.

We consider two types of 1-round algorithms, which as we shall prove, are optimal, each one on its own range of p.

Agreed Meeting Point y (AMP(y)): The algorithm specifies an arbitrary meeting point $y \in [0,1]$, and sets $y_A = y_B = y$. That is, if a player receives a value different than its input value, then it outputs y; otherwise it outputs its input value.

Flip Value (FV): The protocol is the following. If a player receives a value x' different from its input value, then it outputs x'; otherwise it outputs its input value.

Figure 3(a) shows the expected discrepancy of the two algorithms as a function of p, as stated in the following theorem. If $p = q = 1/2$, either AMP or FV can be used, so long as both players use the same algorithm.

▶ **Theorem 3.** *The expected discrepancy of any single-round algorithm is at least* $\min\{q, p^2 + q^2\}$. *This discrepancy is achieved by the AMP algorithm whenever* $p > 1/2$, *and by the FV algorithm whenever* $p < 1/2$ *(and by both if* $p = 1/2$*).*

Proof. For any algorithm, let $\Delta \stackrel{\text{def}}{=} y_B - y_A$, the difference between the output values in an execution. Note that $-1 \le \Delta \le 1$. The adversary selects probabilistically in each round a directed graph representing the correct links from $\mathcal{G} = \{\circ\!\!\leftarrow\!\!\bullet, \circ\!\!\rightarrow\!\!\bullet, \circ\!\!\leftrightarrow\!\!\bullet, \circ\!\!-\!\!\bullet\}$, where the first two graphs are selected with probability pq each, the second with probability p^2 and the fourth with probability q^2. Thus, the expected discrepancy is

$$\begin{aligned}\mathbf{E}[\mathsf{D}] &= (1-0)\cdot q^2 + (1-y_A)\cdot pq + (y_B - 0)\cdot pq + |y_B - y_A|\cdot p^2 \\ &= q^2 + pq(1+\Delta) + p^2|\Delta| \ . \end{aligned} \tag{1}$$

The expected discrepancy is thus a piecewise linear function of Δ. In each "piece," i.e., in each interval of Δ values, $\mathbf{E}[\mathsf{D}]$ attains its smallest value at one of the endpoints of that interval, depending on the sign of $\frac{\mathrm{d}\mathbf{E}[\mathsf{D}]}{\mathrm{d}\Delta}$. We therefore now differentiate Eq. (1) with respect to Δ after conditioning on whether Δ is positive or not. We obtain:

$$\frac{\mathrm{d}\mathbf{E}[\mathsf{D}]}{\mathrm{d}\Delta} = \begin{cases} pq + p^2, & \text{if } 0 \le \Delta \le 1 \\ pq - p^2, & \text{if } -1 \le \Delta < 0 \end{cases} \tag{2}$$

We proceed by case analysis.

- If $\Delta \ge 0$, then $\frac{\mathrm{d}\mathbf{E}[\mathsf{D}]}{\mathrm{d}\Delta} > 0$, and therefore the minimum discrepancy is in the low end of the region $\Delta \ge 0$, i.e., at $\Delta = 0$, where $\mathbf{E}[\mathsf{D}] = q^2 + pq = q$ by Eq. (1).
- If $\Delta < 0$, there are two sub-cases, because, by Eq. (2), $\frac{\mathrm{d}\mathbf{E}[\mathsf{D}]}{\mathrm{d}\Delta}$ is positive if and only if $q > p$.
  - If indeed $q > p$, then the minimum discrepancy is obtained at the smallest possible value of Δ, i.e., when $\Delta = -1$. In this case, by Eq. (1), $\mathbf{E}[\mathsf{D}] = q^2 + p^2$.
  - If $p > q$, then $\frac{\mathrm{d}\mathbf{E}[\mathsf{D}]}{\mathrm{d}\Delta} < 0$ and the minimum discrepancy is obtained at the high end of the region, i.e., at $\Delta = 0$, which we have already analyzed.

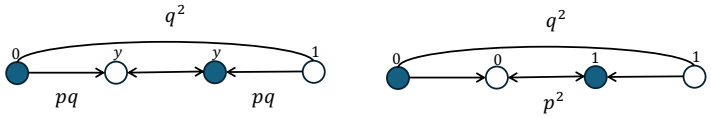
In summary, we have

$$\min \mathbf{E}[\mathsf{D}] = \begin{cases} q & \text{if } p > q, \text{ attained at } \Delta = 0 \\ q^2 + p^2 & \text{if } p < q, \text{ attained at } \Delta = -1 \end{cases} \tag{3}$$

In the case that $p = q$ we get that the minimum discrepancy is $q = q^2 + p^2 = 1/2$ for any $\Delta \in [-1, 1]$.

To see that these are the values achieved by the algorithms AMD and FV, consider Figure 1 (discussed in more detail in Section 4.1), where the four possible executions are depicted as edges with directions indicating that a message is delivered. A vertex corresponds

the the state of each one of the players at the end of the round. Black vertices correspond to Alice and white vertices to Bob, with their decisions on top of the corresponding vertex. For AMP, the undirected edge where no message arrives has discrepancy 1, while the two edges where exactly one message arrives have discrepancy y and $1-y$, respectively. Thus, the expected discrepancy is $q^2+pqy+pq(1-y)$ which is equal to q, independently of the meeting point y. For FV, the bi-directed edge where both messages are delivered has discrepancy 1, while the two other directed edges have discrepancy 0, thus the expected discrepancy is p^2+q^2. ◀



(a) Algorithm AMP **(b)** Algorithm FV

Figure 1 One round protocol complexes for two players starting with a single input assignment. Edges are oriented to indicate which messages arrived during the round, in each execution, and the probability of that execution. On the left are depicted the decisions of the AMP algorithm with meeting point y, on the right are the decisions of algorithm RV.

Assuming at least one message is always delivered, namely, the case when the adversary picks the actual communication graph in the set $\mathcal{G}' = \{\circ\!\!\not\leftarrow\!\!\bullet, \circ\!\!\rightarrow\!\!\bullet, \circ\!\!\not\leftrightarrow\!\!\bullet\}$ has been considered in the past, but for non-deterministic adversaries. For $p = 1/2$, our 1-round algorithms have expected discrepancy $1/3$, which is the optimal discrepancy when the adversary is non-deterministic e.g. [9, 13]. Remarkably, the expected discrepancy improves, both when the probability of message-loss is reduced, and when it is increased, bypassing the $1/3$ lower bound of the non-deterministic case.

We now consider the assumption that it is never the case that no messages arrive in the round. Remarkably, using our algorithms AMP and FV as before, we can see that the expected discrepancy conditioned on the event that at least one message is delivered is at most $1/3$ for any value of p, matching the worst case known bound for non-deterministic adversaries e.g. [9, 13], for two processes. Furthermore, the expected discrepancy improves more and more, both as p gets smaller and smaller than $1/2$, and also as it gets larger than $1/2$, see Figure 2. Formally, we have the following.

▶ **Theorem 4.** *For any single-round algorithm, the expected discrepancy, conditioned on the event that at least one message is delivered, is at least* $\min\left\{\frac{p^2}{1-q^2}, \frac{pq}{1-q^2}\right\}$. *This discrepancy is achieved by the AMP algorithm if* $p > 1/2$, *and by the FV algorithm if* $p < 1/2$, *and by both if* $p = 1/2$, *at a maximum expected discrepancy value of* $1/3$.

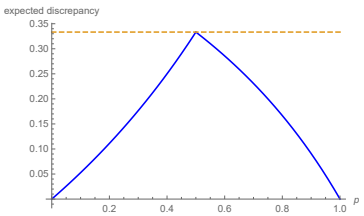


Figure 2 The expected discrepancy of the optimal algorithm, conditioned on the event that not all messages are dropped. The dashed line indicates $1/3$.

3.2 Multiple rounds

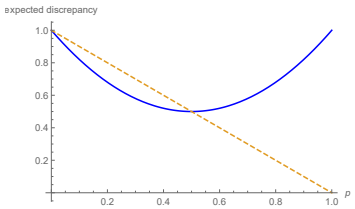
Consider now an algorithm running for k rounds, for any fixed $k \in \mathbb{N}$, known by both processes. Given a single-round algorithm, one simple strategy is *replication*, i.e., sending the 1-round algorithm message k times, so as to decrease the probability that the message is not received from q to q^k. We shall see that the replication strategy is optimal only for $p \geq 1/2$. Instead, we use any of our single-round algorithms *recursively*. That is, we used the output of round $i \geq 1$ as the input of round $i+1$. For simplicity, we can force the inputs to always be in $\{0,1\}$, by using algorithm AMP with meeting point either at 0 or at 1 (algorithm FV satisfies the requirement in any case).

To analyze the expected discrepancy of multiple rounds, we note that the discrepancy of round $i+1$ is a random function (because the execution is stochastic) of a random variable, namely, the discrepancy of round i. The main part of the argument is stated as follows.

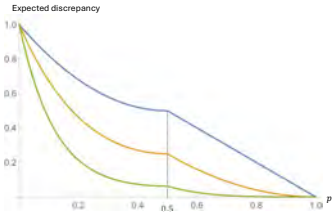
▶ **Lemma 5.** *Let $f_1, f_2, \ldots, f_r$ be a sequence of independently randomized functions, i.e., for every $1 \leq i \leq r$, and every $x \in \mathbb{R}$, $f_i(x)$ is a random variable. Suppose that, for every $i = 1, \ldots, r$, there exists $a_i \in \mathbb{R}$ such that $\mathbf{E}[f_i(x)] \leq a_i x$ for all $x \in \mathbb{R}$. Then $\mathbf{E}[f_r \circ \cdots \circ f_1(x)] \leq \left(\prod_{i=1}^{r} a_i\right)x$. Similarly, if, for every $1 \leq i \leq r$, $\mathbf{E}[f_i(x)] \geq a_i x$ for all $x \in \mathbb{R}$, then $\mathbf{E}[f_r \circ \cdots \circ f_1(x)] \geq \left(\prod_{i=1}^{r} a_i\right)x$.*

▶ **Theorem 6.** *The expected discrepancy of a r-round recursive algorithm is at most q^r if $p \geq 1/2$ by using AMP, and at most $\left(p^2+q^2\right)^r$ if $p < 1/2$ by using FV.*

The result in Theorem 6 is illustrated in Figure 3(b).



(a) The expected discrepancy of the 1-round algorithms as a function of p. The solid line is for the FV rule, and the dashed line is for the AMP rule



(b) The expected discrepancy of recursively applying the optimal 1-round algorithm for 1 round (blue), 2 rounds (orange) and 4 rounds (green) as a function of p

Figure 3 The expected discrepancy of the optimal algorithm.

A recursive algorithm for the case where it is assumed that at least one message is delivered in each round can be derived in a similar way, from Theorem 4 and Lemma 5.

▶ **Theorem 7.** *The expected discrepancy of a r-round recursive algorithm conditioned on the event that at least one message is delivered at each round is at most $\left(\frac{pq}{1-q^2}\right)^r$ if $p \geq 1/2$ by using AMP, and at most $\left(p^2/(1-q^2)\right)^r$ if $p < 1/2$ by using FV. Thus, the expected discrepancy is at most $1/3^r$ for any value of p.*

4 Lower Bound

In this section, we show that the simple two player algorithms consisting of recursively applying AMP or FV as defined in the previous section have the best possible expected discrepancy.

4.1 Protocol Complexes and Algorithms

We already discussed briefly the notion of protocol complex, only for one round, in Figure 1. We explain it in more detail and generalize it to multiple rounds in Section 4.1.1, and then define the notion of an algorithm for a protocol complex in Section 4.1.2.

4.1.1 The Weighted Protocol Complex

We consider the *protocol complex* representation of all executions after r rounds used in the topology perspective of distributed computing [12]. In the case of two players, this *simplicial complex* is merely an undirected graph, but we stick to the terminology for consistency. Without loss of generality, we assume *full information protocols*, that is, every player sends its entire local state at each round. The protocol complex is essentially a representation of the local states of the players in all executions, together with a relation stating which executions are indistinguishable to a process. The local state of a player can be represented as a triple (x_A, x_B, v), where

- x_A and x_B are the binary inputs of Alice and Bob, respectively, where a player sets the input of the other player only once it has received a message from the other, otherwise the value is set $\perp$, and
- v is the *view* of the player, that is, its local state, including the sequence of messages it received (and marking those that it did not receive).

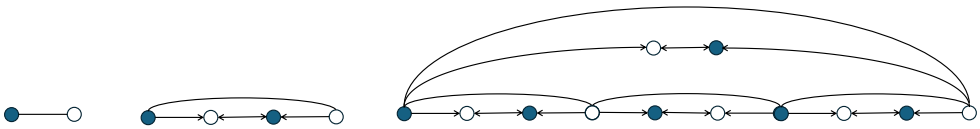
The initial state of Alice (resp., Bob) is thus $(x_A, \perp, v_0)$ (resp. $(\perp, x_B, v_0)$ where x_A (resp., x_B) is the input of Alice (resp., Bob), and $v_0 = ()$ is an empty sequence. An initial configuration thus consists of two initial states, one for Alice and one for Bob. The protocol complex $\mathcal{P}_r$ after r rounds then consists of a graph. Each vertex is a pair (i, s), where $i \in \{A, B\}$ and s is the local state of process i after r rounds, in some execution of the algorithm. Two states (A, s_A), (B, s_B) belong to an edge of the graph, if there is an r round execution, where s_A is the state of A, and s_B is the state of B. Thus the graph is bipartite: each edge connects vertices of different players.

Notice that the protocol complex at round 0 with any *fixed* pair of input values x_A, x_B, consists of a single edge, whose endpoints are the initial states of A and B with those inputs. Figure 4(a) represent such an edge at round 0. The black vertex represents Alice in initial state $(x_A, \perp, v_0)$, and the white vertex represents Bob in initial state $(\perp, x_B, v_0)$. The protocol complex at round 1 depicted on Figure 4(b) is a path of three edges, plus an edge connecting the endpoints of the path. This latter edge depicted on top of the path corresponds to the execution of the (full information) protocol where no message arrives, denoted by $\circ\!\!-\!\!\bullet$. The path of three edges corresponds to the three scenarios $\bullet\!\!\rightarrow\!\!\circ$ (message from Alice reaches Bob), $\bullet\!\!\leftrightarrow\!\!\circ$ (both messages arrive), and $\bullet\!\!\leftarrow\!\!\circ$ (message from Bob reaches Alice). In general, the protocol complex at $r+1$ rounds is obtained from the one at r rounds by replacing each edge by the round 1 protocol complex. Figure 4(c) represents the protocol complex after two rounds, starting from fixed inputs.

Note that, in each round,

$$\Pr(\bullet\!\!\leftrightarrow\!\!\circ) = p^2,\ \Pr(\bullet\!\!\rightarrow\!\!\circ) = \Pr(\bullet\!\!\leftarrow\!\!\circ) = p \cdot q, \text{ and } \Pr(\circ\!\!-\!\!\bullet) = q^2.$$

The probability of an edge in the complex corresponding to a multi-round execution is the product of the probabilities of all the successive scenarios leading to this edge. For instance, the probability of the top edge in Figure 4(c) is q^4 (all messages failed at both rounds), whereas the probability of the edge in the middle of the path at the bottom of Figure 4(c) is p^4 (all messages arrived during both rounds). More generally, the complex that corresponds



(a) Round 0 **(b)** Round 1 **(c)** Round 2

Figure 4 Protocol complexes for two players starting with a single input assignment. Edges are oriented to indicate which messages arrived during the considered round, but the protocol complex is an undirected graph.

to the 0 round execution consists of two nodes and a single edge (see Figure 4(a)), which correspond to the two initial states, and the single possible 0-round (vacuous) execution, whose probability is 1. We denote this complex by G_0. Given a complex $G_i = (V_i, E_i)$, the complex G_{i+1} is obtained by transforming each edge $e \in V_i$ to a 4-node, 4-edge "gadget" denoted $T(e)$, as shown in Figure 5 (recall that the protocol complex in an *undirected* graph).

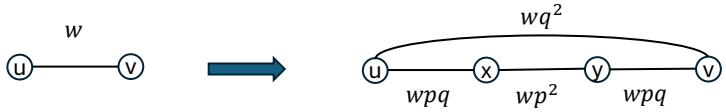


Figure 5 Transformation of an edge by performing one more round. Edge labels indicate their weight.

In our complexes, the edges are weighted: the weight of an edge is the probability of the corresponding execution. Hence the sum of the edge weights must be 1. In particular, the weight of the single edge in G_0 is 1. Given an edge in G_i with weight w, the edges of the gadget corresponding to that edge in G_{i+1} have weights as shown in Figure 5. The rationale is that $T(e)$ corresponds to the executions starting in e with one additional round, in which either both messages are dropped (the (u, v) edge) with probability q^2, or both messages are delivered (the (x, y) edge) with probability p^2, or just one message delivered (the edges (u, x) and (y, v)) with probability pq.

4.1.2 Algorithm for a Protocol Complex

An r-round algorithm for approximate agreement is a deterministic *decision function* δ that assigns a real value $\delta(i, s)$ in $[0, 1]$ to each vertex (i, s), $i \in \{A, B\}$, of the protocol complex at round r, where s is the local state of player i. We denote the decision of process i by $y_i \in [0, 1]$. Remark 2 (Validity) requires that $\delta(i, s) = x_i$, if no message has been received in s, or if a message has been received and the input of the other process is also x_i.

Notice that the decision of a player may depend on the name, A or B, of that player. It is sometime more convenient to write $\delta_i(s)$ for the decision of player i in state s. The algorithm is said to be *symmetric* if δ is solely a function of the local state s, i.e., if the decision functions of the players are equal, $\delta_A = \delta_B$. The *discrepancy* of the execution is $|\delta(A, s) - \delta(B, s)|$, and the expected discrepancy of the algorithm for this input configuration is taken over all possible r-round executions.

As discussed previously, one can assume, without loss of generality, that the input configuration is where A has input 0, and B has input 1.

4.2 Integrality of optimal algorithms

In Section 3.1 we analyzed single round algorithms, and showed that there always exists an optimal algorithm whose only possible output values are 0 and 1. In this section we consider the general case of two player, r-rounds algorithms, and show that the same integrality property holds as well. The idea is the following. For every given input assignment, the discrepancy of an approximate agreement algorithm is a random variable which we wish to minimize. We shall formalize the problem as an optimization problem, which we then convert into a linear program. The result will follow from the fact that the optimal solution is at a vertex of the polytope defined by the constraints.

▶ **Lemma 8** (Integrality Lemma). *There is an optimal algorithm with optimal expected discrepancy in which every player outputs only* 0 *or* 1*.*

Proof. Given an r-round algorithm, each possible execution Γ (specifying which messages are delivered) has a probability $\Pr[\Gamma]$, which can be computed from the model parameters. Since the view at a process, along with its local input, determines the output value of that process, we know that given all inputs and local views, all outputs are determined, and therefore the discrepancy is determined as well. Let $\bar{x} = (x_A, x_B)$ denote the input assignment. There are 4 possible input assignments. If $x_A = x_B$, validity implies that both players output this value, and the discrepancy is 0. We hence focus on the case where $x_A \neq x_B$. Without loss of generality, we assume that $x_A = 0$ and $x_B = 1$. Given an algorithm, let us denote its outputs by $\bar{y} = (y_A, y_B)$. Let Γ^* denote the set of all possible r-round executions for the given input. Given an execution $\Gamma \in \Gamma^*$, let $\Gamma[i]$ denote the view of process i in Γ. Then an algorithm is optimal if its expected discrepancy for the given input has the same value as the solution to the following optimization problem.

$$\text{minimize } \mathbf{E}\left[\mathsf{D}_{\bar{y}}(\bar{x})\right] = \sum_{\Gamma \in \Gamma^*} \Pr[\Gamma] \cdot |y_A(x_A, \Gamma[A]) - y_B(x_B, \Gamma[B])| \tag{4}$$

subject to

$$\forall i \in \{A, B\} \text{ and } \forall \Gamma \in \Gamma^* \ : \min\{x \in \{x_i\} \cup \Gamma[i]\} \leq y_i(x_i, \Gamma[i]) \leq \max\{x \in \{x_i\} \cup \Gamma[i]\} \ . \tag{5}$$

(Abusing notation, we use $\Gamma[i]$ in Eq. (5) to also denote the set of values in the view.)

The constraints force the output values to be within the range spanned by their input value, and the values they received. We stress that the variables are y_A and y_B. The values of x_A and x_B are fixed, and the probabilities are constants whose values are fixed by the stochastic environment model.

Observe that Eq. (4) and (5) do not specify a linear program, due to the absolute values in Eq. (4) – the min and max operators in Eq. (5) are applied to constants as far as the minimization of $\mathbf{E}\left[\mathsf{D}_{\bar{y}}(\bar{x})\right]$ is concerned as, for a given player $i \in \{A, B\}$ and a given execution $\Gamma \in \Gamma^*$, x_i and $\Gamma[i]$ are constants, and so is the smallest (or largest) input value x in $\{x_i\} \cup \Gamma[i]$.

We get around the non-linearity as follows. Consider an optimal solution $\bar{y}$ to the problem specified by Eq. (4) and (5). Let Y denote the multiset of all possible y_i values for the given input, one for each party and each execution. Now, let $z_1, \ldots, z_m$, for an appropriately large m, be the sequence of all values in Y in *increasing order*. If we knew the order of the z values, then we could have replaced the expression $|y_A(x_A, \Gamma[A]) - y_B(x_B, \Gamma[B]|$ with $z_\ell - z_s$, where z_ℓ and z_s are the larger and the smaller values of $\{y_A(x_A, \Gamma[A]), y_B(x_B, \Gamma[B])\}$, respectively. To complete the transformation of the optimization problem defined by Eq. (4)

and (5) into a linear program, we make sure that the z_i variables respect the assumed ordering by adding the constraints

$$z_i \leq z_{i+1} \quad \text{for all } 1 \leq i < m \ . \tag{6}$$

Now, the transformed target function Eq. (4), where a simple subtraction of the z values replaces the absolute value expression in each term, and the constraints Eq. (5) and Eq. (6), where the original y variables replace their z pseudonyms, is a linear program. If the ordering according of the z values agrees with an optimal solution, then the additional constraints of Eq. (6) do not change the target value. In other words, there exists an ordering of the z values such that the optimal algorithm is a solution to the corresponding linear program.

Finally, as mentioned above, if both inputs are the same, the output equals the input value, and the claim is immediate. Otherwise, consider the polytope defined by Eq. (5) using the z variables and Eq. (6). The vertices of this polytope are $\left\{0^\ell 1^{M-\ell} \mid 0 \leq \ell \leq M\right\}$, when the coordinates are ordered according to the z ordering. Since there is always an optimal solution at a vertex, the result follows. ◀

▶ Remark 9. Note that an optimal solution may actually be attained at multiple vertices, in which case all their convex combinations (whose coordinates need not be 0 or 1) are also optimal solutions.

Lemma 8 narrows down the number of candidates for an optimal solution to a finite number of possible algorithms, so in principle one can enumerate all possible 0/1 output values for all inputs and executions to find an optimal solution. But this is hardly practical: The number of possibilities we need to enumerate for a r-round, n-player protocol, even in the case of symmetric algorithms, is doubly-exponential.

4.3 The Lower Bound

We have now all the ingredients to prove a lower bound on the expected discrepancy of any r-round protocol. To this end, fix an algorithm $\mathcal{A}$ and a number of rounds r. We focus on the instance where Alice gets input $x_A = 0$, and Bob gets input $x_B = 1$.

We study the protocol complex $\mathcal{P}$ of $\mathcal{A}$ after r rounds. In this complex, each edge is associated with some probability (depending on the message deliveries in that execution), and a certain discrepancy (depending on the output values defined by the decision function δ of the protocol, in the local states at the edge endpoints). By Lemma 8 we may assume w.l.o.g. that the output values are only 0 and 1. We can therefore partition the vertices of the protocol complex into the set of vertices whose output is 0 and the set of vertices whose output is 1. All edges within a set correspond to executions with discrepancy 0, and all edges that connect vertices in different sets correspond to executions with discrepancy 1. The expected discrepancy is therefore simply the sum of the probabilities of the edges that connect the different sets. In graph theoretic terms, if we view the probability of an execution as the weight of the corresponding edge in the protocol complex, the expected discrepancy is the weight of the edges in the cut defined by the output values. Our goal of obtaining a lower bound on the expected discrepancy can be restated as bounding from below the weight of cuts in the protocol complex.

Let s, t denote the two nodes in the 0-round protocol complex G_0 (see Figure 4(a)). These nodes correspond to the local states of A and B, respectively, where no message is ever received. With these local states, by the validity requirement (Remark 2), A and B are forced to output their input values, namely 0 and 1, respectively. We are thus interested in bounding from below the weight of s-t cuts in the r-round protocol complex G_r.

Let $G = (V, E, w)$ be a graph with edge weights, and let $s, t \in V$ be two distinct nodes in G. Recall that an s-t cut of G, denoted $(S, \bar{S})$ is a partition of V into two subsets S and $\bar{S} = V \setminus S$ where $s \in S$ and $t \in \bar{S}$. The weight of $(S, \bar{S})$, denoted by $w(S; \bar{S})$, is the sum of weights of edges with one endpoint in S and another in $\bar{S}$.

Recall from Section 4.1.1 that $T(e)$ is the protocol complex graph of one round executions starting in e.

▶ **Lemma 10.** *Let $e = \{u, v\}$ be an edge of weight w. Then the weight of the min-weight u-v cut of $T(e)$ is $w \cdot \min\{q, p^2 + q^2\}$.*

Proof. Note that $\min\{q, p^2 + q^2\} = q$ if $p \geq 1/2$ and $\min\{q, p^2 + q^2\} = p^2 + q^2$ if $p \leq 1/2$. Also note that $\min\{q, p^2 + q^2\} < 1$ unless $p = 0$, that is, $\min\{q, p^2 + q^2\} < 1$ unless no message can ever be delivered. We refer to Figure 5 for an illustration. The edge $\{s, t\}$ in $T(e)$ must be in the cut, contributing wq^2 weight. Out of the three edge $\{u, x\}, \{x, y\}$ and $\{y, v\}$ we take an edge of minimal weight, i.e., $\{x, y\}$ if $p < q$ and otherwise either $\{u, x\}$ or $\{y, v\}$, adding a weight of $w \cdot \min\left\{p^2, pq\right\}$. In total, the cut weight is

$$w \cdot \left(q^2 + \min\left\{qp, p^2\right\}\right) \;=\; w \cdot \left(\min\left\{q^2 + qp, q^2 + p^2\right\}\right) \;=\; w \cdot \min\{q, p^2 + q^2\},$$

as claimed. ◀

▶ **Lemma 11.** *Let $i > 0$. If there exists an s-t cut of weight W in G_i then there exists an s-t cut in G_{i-1} of weight at most $W/\min\{q, p^2 + q^2\}$, where s and t are the two nodes of G_0.*

Proof. Let $(S, \bar{S})$ be an s-t cut of G_i. Let $S' = S \cap V_{i-1}$ and $\bar{S}' = \bar{S} \cap V_{i-1}$. Consider the cut $(S', \bar{S}')$ of G_{i-1}. This cut is well defined because $V_{i-1} \subset V_i$. For each edge $e' = \{u, v\}$ that crosses the cut $(S', \bar{S}')$ in G_{i-1} we have that, in G_{i-1} too, $u \in S$ and $v \in \bar{S}$. Therefore some of the edges of $T(e')$ cross the cut $(S, \bar{S})$ as well. By Lemma 10, min-weight cut that separates the endpoints of e' in $T(e')$ have weight $\min\{q, p^2 + q^2\} \cdot w(e')$. Since the edges of different gadgets are disjoint, it follows that

$$w(S', \bar{S}') \;=\; \sum_{e' \in (S', \bar{S})} w(e') \;\leq\; \sum_{e \in (S, \bar{S})} w(e)/\min\{q, p^2 + q^2\} \;\leq\; w(S, \bar{S})/\min\{q, p^2 + q^2\},$$

and the result follows. ◀

We can now establish that our algorithms have optimal expected discrepancy.

▶ **Theorem 12.** *For every $r \geq 1$, every r-round algorithm has expected discrepancy at least $(\min\{q, p^2 + q^2\})^r$.*

Proof. It is sufficient to show that if s, t denote the nodes of the protocol complex G_0, then the weight of any s-t cut in the r-round protocol complex G_r is at least $(\min\{q, p^2 + q^2\})^r$. We proceed by contradiction. If there was a cut of G_r with weight $W < (\min\{q, p^2 + q^2\})^r$, then, by repeated application of Lemma 11, we would infer that there exists an s-t cut of G_0 of weight smaller than $W/(\min\{q, p^2 + q^2\})^r < 1$, which is impossible, as the only s-t cut in G_0 is $(\{s\}, \{t\})$, whose weight is 1 by definition. ◀

5 The case of more than two players

In this section we show that having more than 2 players changes the picture in a profound way. We first show that the integrality lemma does not hold, even for 3 players. We also show that when the number of players is large, 0 discrepancy can be achieved with overwhelming probability in two rounds, or in one round if the number of players with each value is slightly larger than a constant.

Consider a player with input x.

- $p \geq 1/2$: use AMP, with meeting point at $1/2$. I.e., if received $\neg x$ (once or twice), output $1/2$, otherwise output x.
- $\theta \leq p < 1/2$: if received $\neg x$ twice, output $\neg x$; if received $\neg x$ once, output $1/2$; otherwise output x.
- $p < \theta$: if received x (once or twice) or received nothing, output x; otherwise (received no x and at least one $\neg x$), output $\neg x$.

Figure 6 Optimal protocol for three players, given $p \in [0, 1]$. The parameter $\theta \approx 0.35$ is the root of a certain fourth degree polynomial.

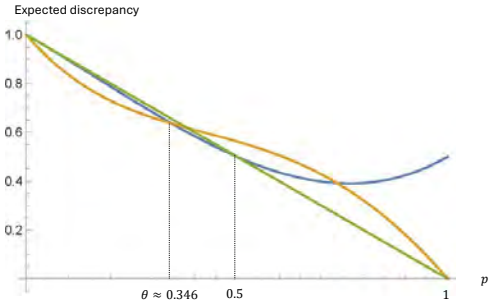


Figure 7 The expected discrepancy of the strategies specified in Figure 6 as a function of p. The green curve is for large p values, the orange for small p values, and the blue for the middle range $\theta < p \leq 1/2$.

5.1 Integrality does not hold for more than 2 players

The linear program of Section 4.2 can be extended to any number of players as follows. While the constraints of Eq. (5) must always be maintained, the target function of Eq. (4) changes. In Eq. (4), we are minimizing the expected discrepancy of a single input assignment. This is fine for the two-party case, because there is only one non-trivial input in this case (namely $\bar{x} = (0, 1)$). However, this is no longer true for more than two players: For example, if there are 3 parties, there are two non-isomorphic nontrivial input assignments: $(0, 0, 1)$ and $(0, 1, 1)$. This means that our goal is to minimize the maximum of the discrepancies of the different inputs.

To deal with this, we introduce a new variable, say v (the goal of the linear program is to minimize v), and a new constraint for each input assignment. Specifically, for each input assignment $\bar{x}$, let $\Gamma(\bar{x})$ denote the set of all executions with input assignment $\bar{x}$, and for any execution Γ, let $\mathsf{D}(\Gamma)$ be the discrepancy of the outputs in Γ. Then, for each input assignment $\bar{x}$, we introduce the constraint $v \geq \sum_{\Gamma \in \Gamma(\bar{x})} \Pr[\Gamma]\mathsf{D}(\Gamma)$ (similarly to the expression in the r.h.s. of Eq. (4).) Since the goal is to minimize v, the program will find the outputs which minimize the worst-case discrepancy, over all inputs.

However, the new constraints change the polytope and create vertices in non-integral coordinates. Indeed, we have computed the optimal 1-round for 3 players using the linear program formulation. The optimal protocol sometimes decides $1/2$: see Figure 6 for a specification of the optimal protocol according to the value of p. Interestingly, the output value $1/2$ is not used when p is small. The expected discrepancy for the three strategies is plotted in Figure 7.

▶ **Theorem 13.** *The optimal expected discrepancy of a single-round algorithm for three players is achieved by the algorithm in Figure 6. The expected discrepancy, plotted in Figure 7, is*

$$\mathbf{E}[\mathsf{D}] = \begin{cases} q\ , & \text{if } p \geq 1/2 \\ \frac{1}{2}(2pq^2 + p^3 + pq(1-q^2) + p^2q^2) + q^2(1-p^2)\ , & \text{if } \theta \leq p < 1/2 \\ p(1-p^2) + q^2(1-pq)\ , & \text{if } 0 \leq p < \theta \end{cases}$$

This discrepancy is achieved by the protocol of Figure 6.

5.2 Asymptotic Analysis as the Number of Players Grows

We have seen that already for three players the optimal expected discrepancy behaves differently than for two players. At the extreme end of the spectrum we have the case of n players, with $n \to \infty$. It turns out that 2 rounds suffice in this case to guarantee 0 discrepancy with high probability. In fact, 0 discrepancy can also be guaranteed w.h.p. in 1 round, if the minority is not too small.

Let us start with the case of one round. We use AMP with any meeting point. Note that the probability that a process does not receive a value decreases exponentially with the number of times that value is sent. Therefore, when one of the values (say 1) is held by a sublogarithmic number of players, there is non-negligible probability that at least one of the other players (with input 0) does not receive any 1 message. Formally, for any $n \geq 2$ and $0 \leq m \leq n$, let

$$I_{n.m} \stackrel{\text{def}}{=} \text{input instance of } n \text{ players, } m \text{ of them with input 0, } n-m \text{ with input 1.}$$

▶ **Theorem 14.** *The expected discrepancy of AMP(a), satisfies* $\mathbf{E}[\mathsf{D}(I_{n.m})] \xrightarrow[n\to\infty]{} 0$*, for any meeting point* $a \in [0,1]$*, when* $\omega(\log_{1/q} n) \leq m \leq n - \omega(\log_{1/q} n)$*.*

Proof. To analyze the behavior of AMP, we use the following shorthand for $I_{n.m}$.

$$A(n,m) \stackrel{\text{def}}{=} \left(1 - q^{n-m}\right)^m\ .$$

Note that $A(n,m)$ is exactly the probability that in $I_{n.m}$, each of the m 0-players received at least one 1 message (from the $n-m$ 1-players).

Note further that for constant $q < 1$, we have that $A(n,m) = \exp\left(-\Theta(mq^{n-m})\right)$. Therefore, for $n - m \geq \omega(\log_{1/q} n)$, i.e., $m \leq n - \omega(\log_{1/q} n)$ we have that $A(n,m) \xrightarrow[n\to\infty]{} 1$.

Now, for Algorithm AMP(a), directly from definitions we have

$$\Pr[\mathsf{D}(I_{n.m}) = 0] = A(n,m)A(n,n-m) \tag{7}$$
$$\Pr[\mathsf{D}(I_{n.m}) = a] = (1 - A(n,m))A(n,n-m) \tag{8}$$
$$\Pr[\mathsf{D}(I_{n.m}) = 1-a] = A(n,m)(1 - A(n,n-m)) \tag{9}$$
$$\Pr[\mathsf{D}(I_{n.m}) = 1] = (1 - A(n,m))(1 - A(n,n-m)) \tag{10}$$

In particular, Eq. (7) implies the probability of consensus approaches 1 when n grows and m is neither too small nor too large. Regarding discrepancy, Eqs. 7–10 imply that the expected discrepancy of AMP(a) is $\mathbf{E}[\mathsf{D}(I_{n.m})] = 1 - \Big(aA(n,m) + (1-a)A(n,n-m)\Big)$. ◀

To cover the case of small minority (of size $O(\log_{1/q} n)$) we "amplify" its presence with another round of communication. Specifically, we have the following algorithm, biased toward output 1.

Algorithm Boost1

1. Round 1: if the local input is 1, send it to all.
2. Round 2: if the local input or any value received in Round 1 is 1, send 1 to all.
3. If the local value is 1 or any value received is 1, decide 1. Otherwise decide 0.

▶ **Theorem 15.** *The expected discrepancy of Algorithm Boost1 approaches* 0 *as the number of players goes to infinity.*

Proof. We prove a stronger statement, namely that the probability that the discrepancy is 0 is $1 - \exp(-\Omega(n))$, where n is the number of players. To see that, note first that if all inputs are 0 the statement is trivial: Boost1 does not send any message in this case, and all processes decide 0. So suppose, without loss of generality, that process 1 has input 1. We show that all other processes decide 1 after 2 rounds, regardless of their initial value. To this end, consider some process i. Process i does not receive the 1 message if for every process j (including $j = 1$ and $j = i$), either the first round message from process 1 to j was dropped, or the send round message from j to i was dropped. Since this happens with probability at most $1 - p^2$ (considering also $j \in \{1, i\}$), we have

$$\Pr[\text{process } i \text{ does not hear "1"} \mid \text{ there exists a "1" input}] \leq (1-p^2)^n \ ,$$

and hence, by the Union Bound,

$$\Pr[\text{there exists a process that does not hear "1"} \mid \text{ there exists a "1" input}] \leq n(1-p^2)^n \ .$$

The discrepancy therefore satisfies $\mathbf{E}[\mathsf{D}] \leq 1 \cdot n(1-p^2)^n \xrightarrow[n\to\infty]{} 0$ for any constant $0 < p \leq 1$. ◀

6 Applications

In this section we present some examples about the use of agreement optimization to solve randomized consensus and approximate agreement.

6.1 Randomized consensus

Consensus in the case of $n = 2$ synchronous reliable processes when messages can be lost is called *coordinated attack* since [10] (also two generals' problem). The *randomized coordinated attack* [22], or for arbitrary number of processes, *randomized consensus*, see, e.g., [2, p.66], for a given desired upper bound error probability $0 < \rho < 1$, is defined by the following conditions, for processes starting with binary input values, and agreeing on binary output values.

Randomized Consensus

Termination: All processes terminate in a bounded number of rounds, r.

Validity: If all processes start with the same value, that is the only value that can be decided.

Randomized agreement: The probability that some process decides 0 and some other process decides 1 is at most ρ.

A straightforward application of Markov-like inequality gives the following result, for any number of players.

▶ **Theorem 16.** *Let $\mathcal{A}$ be an algorithm for agreement optimization with possible output discrepancy values $v_0 = 0 < v_1 < \ldots < 1$. If the expected discrepancy of $\mathcal{A}$ is μ, then $\mathcal{A}$ solves randomized consensus with error probability $\rho \leq \frac{\mu}{v_1}$.*

Since our 2-party algorithms produce discrepancy either 0 or 1, we have the following result.

▶ **Corollary 17.** *For any given $r \in \mathbb{N}$, two-player randomized consensus can be solved in r rounds with error probability at most $\min(p^2+q^2, q)^r$. Moreover, no protocol can solve randomized consensus in r rounds with probability larger than $1 - \min(p^2+q^2, q)^r$.*

6.2 Approximate Agreement

The usual ϵ-approximate agreement problem requires processes to decide values at most ϵ apart. Here we define a relaxed version, allowing an error of ρ. Formally, it requires the following to hold.

(ϵ, ρ)-Approximate Agreement
Termination: All processes terminate in a bounded number of rounds, r.
Validity: The value decided by every processor is in the range spanned by all input values.
Randomized ϵ-Agreement: The probability that the discrepancy is larger than ϵ is at most ρ.

In the classic version of approximate agreement [6], the input values are arbitrary real numbers. Input values in a bounded region such as $[0, 1]$, have also been considered e.g. [19]. The number of rounds r depends on ϵ. In fact, it may depend also on the discrepancy of the input values (the maximal distance between inputs) or even their magnitude, see e.g. [3]. The *relative discrepancy* agreement optimization can be defined as its discrepancy normalized by the discrepancy of the input values.

We have the following consequence of our lower bound on the expected discrepancy of agreement optimization. It is stated for inputs in $[0, 1]$ (it could be rephrased in terms of the discrepancy of the inputs)

▶ **Theorem 18.** *Any algorithm for two player (ϵ, ρ)-Approximate Agreement that terminates in r rounds when the inputs are binary must have probability of error of at least $\rho \geq \frac{(\min(p^2+q^2,q))^r - \epsilon}{1-\epsilon}$.*

We can also derive upper bounds on (ϵ, ρ)-Approximate Agreement using agreement optimization, for the case of 2 players with arbitrary real input values, as follows.

First, note that non-binary inputs may be a problem for algorithms such as AMP: where should the meeting point be located? If we fix the location in advance, validity may be violated. And if a process computes it based on the values it has seen, the meeting point may be different at different processes because their views may be different. However, this is easily solvable in the case of two processes. The point is that the agreed meeting point is used only when two values are known, and in the case of two processes, if both their views contain two values, they must contain the same two values, so it is possible to specify a common meeting point.

Specifically, consider the following algorithm for $a \in [0, 1]$.

Algorithm Scaled AMP(a) (2 players, arbitrary input values in $\mathbb{R}$)
- If no value is received, output the input value.
- Otherwise, denote the local input and the 4 value by x and y. Output $\min\{x, y\} + a \cdot |y - x|$.

Clearly, scaled AMP is a generalization of AMP to any input values, assuming there are only two players. Since Algorithm FV also works with arbitrary input values, we arrive at the following result.

▶ **Theorem 19.** *Let $0 < \rho < 1$. Assume that $p > 1/2$. Then 2-player (ϵ, ρ)-Approximate Agreement can be solved with probability $1 - \rho$ in $O(\log 1/\rho)$ rounds with relative discrepancy $\epsilon = q^{\Omega(\log(1/\rho))}$.*

We note that we have not attempted to optimize the constants. For the case of $p \leq 1/2$, we use the recursive FV algorithm and the following similar statement holds.

▶ **Theorem 20.** *Let $\rho > 0$. Assume that $p \leq 1/2$. Then 2-player (ϵ, ρ)-Approximate Agreement can be solved in $O(\log(1/\rho))$ rounds with relative discrepancy $\epsilon = (p^2+q^2)^{\Omega(\ln(1/\rho))}$.*

The nice property of scaled AMP is that it allows us to use AMP(a) with $0 < a < 1$ in the recursive version, thus obtaining a "continuous" algorithm, i.e., an algorithm in which the discrepancy shrinks (probabilistically) in each round, rather than algorithms in which the discrepancy is either 0 or 1.

▶ **Theorem 21.** *The optimal relative discrepancy of 2-party approximate agreement by a r-round algorithm is $\min(q, (p^2 + q^2))^r$, achieved by r recursive applications of algorithm FV if $p \leq 1/2$ of scaled AMP(a) (with any $0 \leq a \leq 1$) for $p \geq 1/2$.*

7 Conclusion

In this paper we have considered a stochastic dynamic network model consisting of synchronous, reliable processes communicating through channels that may drop messages with a given probability p. We defined the agreement optimization problem, where the goal is to minimize the expected discrepancy. For the case of two players, we provided a detailed characterization of the achievable expected discrepancy. To this end we developed the Integrality Lemma, showing that it is sufficient to consider algorithms with outputs in $\{0, 1\}$. Going beyond $n = 2$, we showed that the problem becomes much more complicated: this lemma no longer holds, and optimal algorithms may have more different behaviors depending on the parameter p, while in the case of two players, there are only two possible behaviors, for p either smaller or larger than $1/2$. We also presented algorithms for large number of processes, whose expected discrepancy goes down to 0 exponentially fast. Finally, we showed the relevance of the agreement optimization problem to obtain algorithms for randomized consensus and randomized approximate agreement, with small probability of error.

We leave many interesting open problems for future work. A main question is to find optimal algorithms for agreement optimization for any number of processes $n > 2$. Our analysis of agreement optimization is mostly under the simplest stochastic assumption, with a single parameter p, known to all processes. The full version extends some of our results the case where the parameter may be different in different links or rounds, but there are many other stochastic models that have been considered in the past, as mentioned in the Introduction, for which agreement optimization has not been studied, e.g. a model where the links are unidirectional (and each one fails with probability p).

The analysis of our algorithms was performed by considering a *weighted* version of the protocol complex of the topology approach to distributed computing, which is, up to our knowledge, new. It would be interesting to extend it to $n > 2$ processes.

We have considered binary consensus and 1-dimensional approximate agreement. It would be interesting to consider multi-valued versions of consensus, e.g., [18], and multi-dimensional versions of approximate agreement [15]. Regarding other tasks, several have already been studied in dynamic networks, such as k-consensus [17] and set agreement [4, 8], but not in our stochastic setting.

References

1 Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. *Distributed Comput.*, 32(6):505–515, 2019. doi:10.1007/S00446-017-0303-5.

2 James Aspnes. Notes on theory of distributed systems, 2023. arXiv:2001.04235.

3 Hagit Attiya and Faith Ellen. The Step Complexity of Multidimensional Approximate Agreement. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2022.6.

4 Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theor. Comput. Sci.*, 726:41–77, 2018. doi:10.1016/J.TCS.2018.02.019.

5 Michael Dinitz, Jeremy Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of information spreading in dynamic networks. *J. ACM*, 71(3), June 2024. doi:10.1145/3661831.

6 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, May 1986. doi:10.1145/5925.5931.

7 Antoine El-Hayek, Monika Henzinger, and Stefan Schmid. Broadcast and Consensus in Stochastic Dynamic Networks with Byzantine Nodes and Adversarial Edges. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.21.

8 Pierre Fraigniaud, Minh Hang Nguyen, and Ami Paz. *A Simple Lower Bound for Set Agreement in Dynamic Networks*, pages 253–262. SIAM, 2025. 2025 Symposium on Simplicity in Algorithms (SOSA). doi:10.1137/1.9781611978315.20.

9 Matthias Függer, Thomas Nowak, and Manfred Schwarz. Tight bounds for asymptotic and approximate consensus. *J. ACM*, 68(6), October 2021. doi:10.1145/3485242.

10 Jim Gray. Notes on data base operating systems. In Michael J. Flynn, Jim Gray, Anita K. Jones, Klaus Lagally, Holger Opderbeck, Gerald J. Popek, Brian Randell, Jerome H. Saltzer, and Hans-Rüdiger Wiehle, editors, *Operating Systems, An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer, 1978. doi:10.1007/3-540-08755-9_9.

11 Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. doi:10.1002/net.3230180406.

12 Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013.

13 Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006. doi:10.1137/S0097539701397412.

14 Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *SIGACT News*, 42(1):82–96, 2011. doi:10.1145/1959045.1959064.

15 Hammurabi Mendes, Maurice Herlihy, Nitin H. Vaidya, and Vijay K. Garg. Multidimensional agreement in byzantine systems. *Distributed Comput.*, 28(6):423–441, 2015. doi:10.1007/S00446-014-0240-5.

16 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Commun. ACM*, 61(2):72, January 2018. doi:10.1145/3156693.

17 Henrique Moniz, Nuno Ferreira Neves, Miguel Correia, and Paulo Veríssimo. Randomization can be a healer: consensus with dynamic omission failures. *Distributed Computing*, 24(3):165–175, 2011. doi:10.1007/s00446-010-0116-2.

18 Achour Mostéfaoui, Michel Raynal, and Frederic Tronel. From binary consensus to multivalued consensus in asynchronous message-passing systems. *Inf. Process. Lett.*, 73(5-6):207–212, 2000. doi:10.1016/S0020-0190(00)00027-2.

19 E. Schenk. Faster approximate agreement with multi-writer registers. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 714–723, 1995. doi:10.1109/SFCS.1995.492673.

20 Ulrich Schmid. Failure model coverage under transient link failures. Technical report, Research Report 2/2004, Technische Universität Wien, Institut für Technische Informatik, 2004, 2008. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=35b27320d6e1c34ef233f178411d20973d7813f5.

21 Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912–1951, 2009. doi:10.1137/S009753970443999X.

22 George Varghese and Nancy A. Lynch. A tradeoff between safety and liveness for randomized coordinated attack. *Information and Computation*, 128(1):57–71, 1996. doi:10.1006/inco.1996.0063.

Approach of Agents with Restricted Fuel Tanks

Adam Ganczorz
Institute of Computer Science, University of Wrocław, Poland

Tomasz Jurdzinski
Institute of Computer Science, University of Wrocław, Poland

Andrzej Pelc
Département d'informatique, Université du Québec en Outaouais, Gatineau, Canada

Grzegorz Stachowiak
Institute of Computer Science, University of Wrocław, Poland

Abstract

Two mobile agents, modelled as points in the plane moving at speed 1, have to get at a distance at most 1 from each other. This task is known as approach or rendezvous in the plane. An adversary initially places both agents at distinct points, called their bases, at distance at most D, and wakes them up at possibly different times. Each of the agents has a fuel tank that allows them to traverse a trajectory of length D, and can be replenished at the base of the agent. The algorithm of each agent consists of a series of actions which are either moves at a chosen distance in a chosen direction or staying idle for a chosen period of time. For a given instance of the approach task, the execution time of an approach algorithm is the length of the period between the start of the later agent and the moment of approach.

Our goal is to design approach algorithms with optimal time complexity. We consider two independent *coherence assumptions*. One of them is *time coherence*, i.e., agents start simultaneously, and the other is *orientation coherence*: agents have compatible compasses, showing the same North direction. Our main result is establishing optimal time complexity of the approach problem with restricted fuel tanks. It turns out that this optimal complexity heavily depends on the above coherence assumptions. If both of them are satisfied then approach can be performed in time $O(D^2)$ and we show that this complexity is optimal. If any of the two coherence assumptions is missing then approach can be performed in time $O(D^2\sqrt{D})$ and we prove that this order of magnitude cannot be improved.

Our main technical contribution are lower bounds showing that, for each of the considered scenarios, our fairly natural approach algorithms are, in fact, optimal.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases mobile agent, approach, rendezvous, plane, restricted energy

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.33

Funding Supported by the National Science Center, Poland (NCN), grant 2020/39/B/ST6/03288. *Andrzej Pelc*: Partially supported by NSERC discovery grant RGPIN 2024-03767 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

1.1 The background

Two mobile agents, modelled as points moving in the plane, have to get at a distance at most 1 from each other. This task is known as approach or rendezvous in the plane and has numerous applications. The final distance, normalized to 1, should be viewed as the distance at which agents can "see" each other, hence the goal of approach is mutual perception. In human interaction, people may want to meet in a large terrain, where meeting means seeing each other. In robotics applications, two mobile robots, independently deployed in a


© Adam Ganczorz, Tomasz Jurdzinski, Andrzej Pelc, and Grzegorz Stachowiak;
licensed under Creative Commons License CC-BY 4.0



contaminated territory and collecting samples of the ground, may need to meet in order to exchange these sample and coordinate further actions. Robots have limited energy, either a fuel tank or an electric battery, that allows them to travel a certain maximum distance and then has to be replenished at the base.

1.2 The model and the problem

We consider two mobile agents modelled as points moving in the plane, that have to get at a distance at most 1 from each other. An adversary initially places both agents at distinct points, called their bases, at distance at most D, and wakes them up at possibly different times. Both agents execute the same deterministic algorithm. If they were identical then, in the case of simultaneous start and identical compasses, they would move along trajectories that are shifts of each other, and consequently, at any time, their distance would be the same as at the start, thus precluding approach, for any $D > 1$. In order to allow approach, this symmetry has to be broken. We do this in the way that is most common in the literature: agents have distinct labels that are integers from a set $\{1, \ldots, L\}$, where L is some constant number. This may be viewed as the set of all identifiers used by the manufacturer of the robots. We make an assumption, standard in rendezvous/approach literature (cf. e.g., [4, 10]) that each agent only knows its own label that can be used as a parameter in a common deterministic algorithm they execute. Each agent is equipped with a clock and a compass showing the cardinal directions. Clocks of the agents tick at the same rate and the clock of each agent starts at its wake-up. Compasses may be inaccurate and arbitrarily distorted, i.e., for each agent, its North can point in any (fixed) direction. Each of the agents has a fuel tank that allows them to traverse a trajectory of length D, and can be replenished at the base of the agent.[1]

The execution of an approach algorithm by an agent consists of a series of actions. An agent can either choose a direction α (according to its compass) and a distance x, in which case it travels distance x in direction α with speed 1, or it can stay put at the current point for a chosen time t. Whenever agents get at a distance at most 1, the algorithm is interrupted and the goal of approach is achieved. For a given instance of the approach task, the execution time of an approach algorithm is the length of the period between the start of the later agent and the moment of approach.

1.3 Our results

Our goal is to design approach algorithms with optimal time complexity. We consider two independent *coherence assumptions*. One of them is *time coherence*, i.e., agents start simultaneously, and the other is *orientation coherence*: agents have compatible compasses, showing the same North direction. The presence or absence of these two assumptions yields four possible scenarios. Our main result is establishing optimal time complexity of the approach problem for each of these scenarios. It turns out that this optimal complexity heavily depends on the above coherence assumptions. If both of them are satisfied then approach can be performed in time $O(D^2)$ and we show that this complexity is optimal. If any of the two coherence assumptions is missing then approach can be performed in time $O(D^2\sqrt{D})$ and we prove that this order of magnitude cannot be improved.

Our main technical contribution are lower bounds showing that, for each of the considered scenarios, our fairly natural approach algorithms are, in fact, optimal. In both our $\Omega(D^2\sqrt{D})$ lower bound proofs, we consider the most difficult instances of the approach problem, when

[1] The discussion of the size of the tank is postponed to Section 5.

bases of the agents are at a distance exactly D. For these instances, approach is only possible in short time periods in which both agents are close to their maximal range, i.e., at distance $D/2$ from their respective bases. We will prove that these time periods must be separated by time segments of length $\Omega(D)$. Due to lack of simultaneous start or of compass compatibility, any of these time periods may be necessary for approach, depending on the instance. In view of the above, the main difficulty is proving that the number of these time periods is $\Omega(D\sqrt{D})$, which will imply the lower bound $\Omega(D^2\sqrt{D})$ on the time of approach.

1.4 Related work

The task of rendezvous of mobile agents has been extensively studied, both when agents move in graphs, and when they move in the plane. For a survey of randomized rendezvous algorithms, see the classic book [1], amd for a survey of deterministic rendezvous, see [16].

Deterministic rendezvous in networks modelled as graphs has been studied under two scenarios. In the synchronous scenario, agents move in synchronous rounds, in each round deciding either to stay put at the current node or to move to an adjacent neighbor. The goal is to get both agents at the same node in the same round. The time of rendezvous is the number of rounds used. A time-efficient synchronous rendezvous algorithm was designed in [17], although the problem of optimal synchronous rendezvous is still open. In the asynchronous scenario, an agent chooses the edge along which it wants to go but the adversary chooses the speed of the agent during the move. The goal is to meet at a node or inside an edge, and the cost is the number of traversed edges. A polynomial-cost algorithm for asynchronous rendezvous in arbitrary graphs was designed in [10].

Rendezvous in the plane, often called approach, has also been studied both under the synchronous and asynchronous scenarios. In [8, 9], this task was investigated under various assumptions concerning attributes of agents, such as orientation, chirality and speed. An asynchronous approach algorithm in the plane working at cost polynomial in the initial distance and in the lengths of agents' labels was designed in [4]. The more general task of gathering many agents in the plane was studied in [6, 7, 12], under various assumptions concerning perception capabilities of the agents.

A special, well researched variation of rendezvous, both in graphs and in the plane, is treasure hunt. An inert treasure is hidden at a node of the graph or in a point in the plane, and a mobile agent starting at some other node or point in the plane has to find it. Thus, treasure hunt can be viewed as rendezvous in which one of the agents is permanently inert. Treasure hunt in the plane was considered, e.g., in [3, 15], see also surveys [13, 14]. Treasure hunt in graphs was investigated, e.g., in [2, 5, 11], under some constraints imposed on the mobile agent: in [2], the mobile agent had a restricted fuel tank that could be replenished at the base (as in the present paper), and in [11], the agent was tethered, i.e., attached to the base with a rope of fixed length. To the best of our knowledge, the general task of approach in the plane of agents with restricted fuel tanks has never been studied before.

2 Preliminaries

We will use the following geometric lemma which informally says that two points on tangent circles C_1 and C_2 of radius r located roughly symmetrically with respect to the tangent line of C_1 and C_2 are at a distance at most 1, provided that they are in arc-distance $O(\sqrt{r})$ from the common point of C_1 and C_2. The proof of Lemma 1 is in the Appendix, Section A.

▶ **Lemma 1.** *Let C_1 and C_2 be tangent circles of the same radius r, centered at O_1 and O_2, respectively. Let S be the common point of C_1 and C_2. For $i = 1, 2$, and for any point P on C_i, let $\alpha_i(P)$ be the length of the arc PS on C_i. Let $y = \sqrt{r}/4$. Let P_1 be a point on C_1 such that $\alpha(P_1) \leq y$, and let P_2 be a point on C_2 such that $\alpha_1(P_1) - 1/2 \leq \alpha_2(P_2) \leq \alpha_1(P_1) + 1/2$. Suppose that either P_1 on C_1 is counterclockwise from S and P_2 on C_2 is clockwise from S, or P_1 on C_1 is clockwise from S and P_2 on C_2 is counterclockwise from S. Then $|P_1P_2| < 1$. (See Figure 1.)*

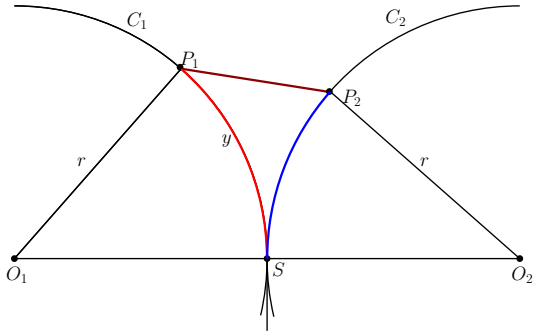


Figure 1 Illustration for Lemma 1. The lemma says that, if the arc length of P_1S is at most $\sqrt{r}/4$, this length and the length of P_2S differ by at most $1/2$, then the distance between P_1 and P_2 is less than 1.

3 Algorithms

This section is devoted to our positive results. We design two approach algorithms: an algorithm with time complexity $O(D^2)$ for the strongest scenario where both coherence assumptions are satisfied, and an algorithm with time complexity $O(D^2\sqrt{D})$ working even in the weakest scenario where none of the coherence assumptions is satisfied.

For both algorithms, two variations are presented: one for agent Blue and the other for agent Red. This corresponds to a simplified version of our model, in which agents have labels 1 and 2, i.e., they know from the outset how symmetry between them is broken (they know which of them is Blue and which is Red). In Section 3.3, we show how these algorithms can be modified in order to conform with our model, where agents have distinct labels from the set $\{1, \ldots, L\}$ and each agent only knows its label, i.e., it does not know if it is Blue or Red.

3.1 Time $O(D^2)$ for simultaneous start and common orientation

In this section, we design Algorithm `Faster` working in the strongest scenario where both coherence assumptions are satisfied. We prove that this algorithm has complexity $O(D^2)$.

Let O_1 be the base of agent Blue and O_2 the base of agent Red. Let C_1 be the circle of radius $D/2$ centered at O_1 and let C_2 be the circle of radius $D/2$ centered at O_2. Let $n = 4\lceil \pi D \rceil$. Let $A_0, \ldots, A_{n-1}$ (resp. $B_0, \ldots, B_{n-1}$) be points on C_1 (resp. on C_2) in clockwise order, such that A_0 is North of O_1 B_0 is North of O_2, and all distances between points A_i and $A_{(i+1) \mod n}$ and points B_i and $B_{(i+1) \mod n}$ are equal. Each agent visits consecutive points on its circle but they start from antipodal points.

Algorithm `Faster` for agent Blue (Red, resp.): the algorithm works in n phases. In phase $i = 0, 1, \ldots n-1$, agent Blue (Red, resp.) goes from its base straight to A_i ($B_{(i+n/2) \mod n}$, resp.) and comes back straight to its base.

The following theorem establishes the correctness and complexity of Algorithm `Faster`. Its proof is in the Appendix, Section B.

▶ **Theorem 2.** *Algorithm* `Faster` *guarantees approach in time* $O(D^2)$.

3.2 Time $O(D^2\sqrt{D})$ for arbitrary start delay and arbitrary orientations

In this section, we design Algorithm `Slower` working even in the weakest scenario where none of the coherence assumptions is satisfied. We prove that this algorithm has complexity $O(D^2\sqrt{D})$. Let O_1 be the base of agent Blue and O_2 the base of agent Red. Let C_1 be the circle of radius $D/2$ centered at O_1 and let C_2 be the circle of radius $D/2$ centered at O_2. Let $n = \lceil 2\pi\sqrt{D} \rceil$ and let $m = \lceil 4\pi D \rceil$. Let $A_0, \ldots, A_{n-1}$ (resp. $B_0, \ldots, B_{m-1}$) be points on C_1 (resp. on C_2) in clockwise order, such that A_0 is North of O_1 according to agent's Blue compass, B_0 is North of O_2 according to agent's Red compass, and all distances between points A_i and $A_{(i+1) \mod n}$ and points B_i and $B_{(i+1) \mod n}$ are equal. Each agent visits consecutive points on its circle. However, agent Blue visits only $\Theta(\sqrt{D})$ points on C, waiting in each of them for time $\Theta(D^2)$, while agent Red visits $\Theta(D)$ points on C without any waiting.

Algorithm `Slower` for agent Blue: The algorithm works in phases. In phase $i \geq 0$, agent Blue goes from its base straight to $A_{i \mod n}$, waits there for time $4\lceil 4\pi D^2 \rceil$, and comes back straight to its base.
Algorithm `Slower` for agent Red: The algorithm works in phases. In phase $i \geq 0$, agent Red goes from its base straight to $B_{i \mod m}$ and comes back straight to its base.

The following result establishes the correctness and complexity of Algorithm `Slower`. Its proof is in the Appendix, Section C.

▶ **Theorem 3.** *Algorithm* `Slower` *guarantees approach in time* $O(D^2\sqrt{D})$.

3.3 Algorithms for agents with arbitrary labels

The algorithms presented so far assume that each agent knows in advance whether it is Blue or Red, and therefore it can execute the appropriate version of the algorithm. However, in our model we assume that the agents do not have prior knowledge of their roles. The only way to break symmetry is provided by distinct labels of agents from the set of natural numbers $\{1, \ldots, L\}$, where the value of L is a constant not known to the agents. Each agent knows only its own label that can be used as a parameter in their common deterministic algorithm.

It turns out that our algorithms can be adjusted to the above scenario at the multiplicative cost in time complexity of the order of $O(\log L)$. Since we assume that L is a constant, this does not change the previously established complexities. The idea behind the generalizations is that the original algorithms presented earlier are executed by each agent several times, until the approach happens. The agent uses an appropriate binary encoding of its label in order to decide whether to act as Blue or Red in its ith execution. More precisely, the agent X with the binary encoding of its label equal to $l(X)$ executes the algorithm for Blue in the ith execution if the ith bit of $l(X)$ is equal to 0 and it executes the algorithm for Red otherwise. Using fairly standard encodings of size $O(\log L)$ and appropriate proof techniques one can prove that there will appear simultaneously a period in which one of the agents acts as Blue and the other one acts as Red in the first $O(\log L)$ executions of the original algorithm designed for agents Blue and Red. These facts will lead to the following result.

▶ **Theorem 4.** *There exist algorithms for agents with unique labels from the set* $\{1, \ldots, L\}$ *which:*

- *guarantee approach in time* $O(D^2 \log L)$ *in the model with common orientation and simultaneous start (using the assumption that* L *is constant, this complexity is* $O(D^2)$*);*
- *guarantee approach in time* $O(D^2\sqrt{D} \log L)$ *in the model without simultaneous start and without common orientation (using the assumption that* L *is constant, this complexity is* $O(D^2)$*).*

4 Lower bounds

This section is devoted to our negative results. We present three lower bounds. The first is the lower bound $\Omega(D^2)$ on the time approach that holds for the strongest scenario with simultaneous start and common orientation. This lower bound shows that our algorithm for this scenario has optimal time complexity. The two other lower bounds concern scenarios when one of the coherence assumptions is not satisfied. If either arbitrary delay between starting times is permitted or different orientations of agents are allowed, we prove that approach requires time $\Omega(D^2\sqrt{D})$. This result, combined with the complexity $O(D^2\sqrt{D})$ of our algorithm working even in the weakest scenario shows that $\Theta(D^2\sqrt{D})$ is the optimal complexity of approach in all the scenarios except the strongest one.

4.1 Time $\Omega(D^2)$ for simultaneous start and common orientation

As a warm-up, in this section, we prove the lower bound $\Omega(D^2)$ on the time of approach that holds even for the strongest scenario with simultaneous start and common orientation. In fact we will show the following stronger lower bound implying our result: it turns out that this lower bound holds even for unbounded fuel tanks. Its proof is in the Appendix, Section D.

▶ **Theorem 5.** *The time of approach for two agents simultaneously starting at a distance at most* D*, with common orientation and unbounded tanks is* $\Omega(D^2)$*.*

Proof. Consider an arbitrary approach algorithm for two agents A and B. Let $a(t)$ (resp. $b(t)$) be a vector describing the position of agent A (resp. B) relative to its starting point in time t. Let $d(t) = a(t) - b(t)$. Let T be the approach time of the algorithm.

For $i, j \in [-\lfloor \frac{D}{4} \rfloor, \lfloor \frac{D}{4} \rfloor - 2]$ let $r_{i,j}$ be the rectangle with vertices in points $(2i, 2j), (2i + 2, 2j), (2i, 2j + 2), (2i + 2, 2j + 2)$.

We will consider only *even* rectangles, that is the rectangles $r_{i,j}$ where both i and j are even. We say that the algorithm *marks the rectangle* $r_{i,j}$ if there is a moment t of the execution of the algorithm such that the point $d(t)$ is inside $r_{i,j}$. As for any t_1, t_2 such that, $|t_1 - t_2| \leq 1$ the length of the vector $d(t_1) - d(t_2)$ is at most 2, the algorithm can mark at most $T + 1$ even rectangles.

Assume that the time of approach T is less than $\frac{D^2}{16}$. The algorithm can mark at most $\frac{D^2}{16} + 1$ even rectangles, and thus there exists an unmarked even rectangle $r_{i,j}$. By fixing the base of agent A at point $(0, 0)$ and the base of B at point $(2i + 1, 2j + 1)$ the adversary can guarantee that the approach does not occur. This contradiction concludes the proof. ◀

4.2 Geometric preliminaries for $\Omega(D^2\sqrt{D})$ lower bounds

In this section we define the common notions and prove geometric properties useful in lower bounds from Sections 4.4 and 4.5. Some missing proofs will be provided in the full version of the paper.

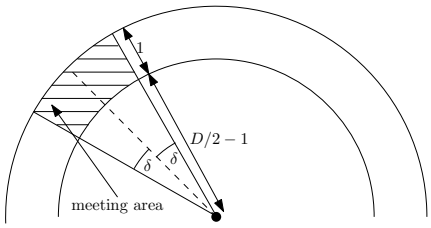


Figure 2 The meeting area of a walk: The dashed line is the walk bisector. The marked area is the meeting area, where $\delta = \arcsin(\frac{1}{D/2-1})$.

Consider an arbitrary approach algorithm for two agents A and B. Each agent's *range disc* is the disc with radius $D/2$ centered at the base of the agent. Each agent's *range circle* is the boundary of its range disc. We assume that the bases of the agents are at distance D, i.e., their range discs are tangent. Consider the trajectory of an agent between its wake-up and the approach. This trajectory can be partitioned into a sequence of sub-trajectories between consecutive visits of the base, and the final sub-trajectory finishing at the approach. These sub-trajectories are called *walks*. For each walk, we distinguish its *meeting part* between the first and last points of the walk at distance at least $D/2-1$ from the source. The bisector of the angle between these points is called the *walk bisector*. The angle between North and the walk bisector of agent A is called the *walk angle* of agent A. The angle between South and the walk bisector of agent B is called the *walk angle* of agent B. We define the angle $\delta = \arcsin(\frac{1}{D/2-1})$. The *meeting area of a walk* is the part of the plane bounded by two half-lines with origin at the base, forming angle δ with the walk bisector, and circles of radii $D/2$ and $D/2-1$ centered at the base (see Fig. 2).

▶ **Fact 1.** *If an agent goes back to its base at the end of a walk w, the intersection of w with the annulus between circles of radii $D/2-1$ and D centered at the base of this agent is included in the meeting area of the walk w.*

Proof. In order to get back to the base after leaving the disk Δ with the radius $D/2-1$ centered at the base, an agent can only move at distance at most 1 before getting back to the disc Δ. A possible part of the trajectory outside of Δ corresponds to an angle $\arcsin(\frac{1}{D/2-1})$. ◀

Note that approach can occur only when an agent is in the meeting area of its walk.

Consider a *geometric instance* I of the approach problem consisting of bases a of agent A and b of agent B. The angle between the segment ab and the half-line with direction North, originating at a is denoted by $\gamma(I)$. This is also the angle between the segment ab and the half-line with direction South, originating at b.

We define a *pair of walks* as a two-element set containing a walk of agent Blue and a walk of agent Red. We say that a pair of walks $\{w_1, w_2\}$ *supports* the geometric instance I if these walks are at distance at most 1.

Now, we provide some properties regarding "usefulness" of walks and pairs of walks with respect to the success of approach (cf. Fig. 3).

In the following lemma we focus on a single walk. We show that, a given walk w can belong to a pair of walks assuring approach for an instance I, only if the walk bisector of w is $O(1/\sqrt{D})$-close to $\gamma(I)$. In other words, the angle between the segment connecting the bases and the walk bisector of w should be $O(1/\sqrt{D})$. Otherwise, the agent is at distance greater than 1 from any point reachable by the other agent during the walk w of the first agent. The proof of the lemma is in the Appendix, Section E.

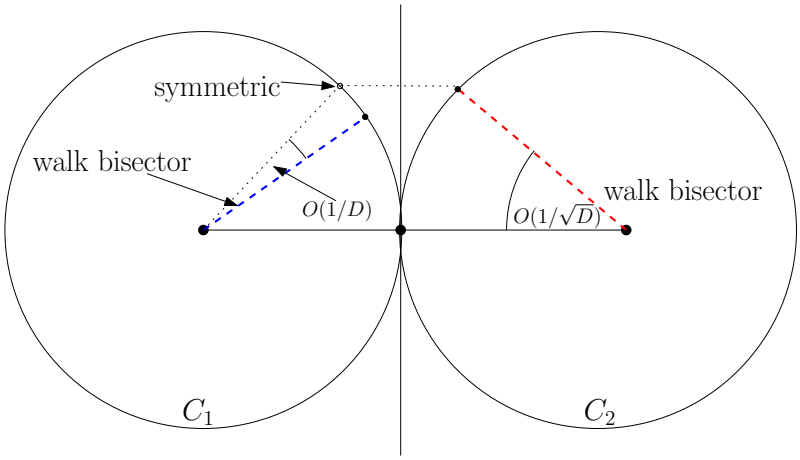


Figure 3 Illustration for Lemmas 6 and 7. The blue dashed segment is the walk bisector of agent Blue and the red dashed segment is the walk bisector of agent Red.

▶ **Lemma 6.** *If $|\alpha - \gamma(I)| > 3/\sqrt{D}$, for $D > 256$, then no pair of walks $\{w_1, w_2\}$ of agents A and B, such that α is the walk angle of one of the agents, supports the geometric instance I.*

Now, we analyze additional requirements which have to be satisfied for a pair of walks in order to make approach feasible, provided that walks from the pair are appropriately synchronized in time and both of them are close to the tangency point of their circles with radius $D/2$, as expressed in Lemma 6. In Lemma 7 we show that, apart from the requirements of Lemma 6, a pair of walks $\{w_1, w_2\}$ might lead to approach only if their walk bisectors are $O(1/D)$-close. More precisely, if α and β are the angles of bisectors of these walks (clockwise from Blue and counterclockwise from Red) then approach might be possible only if $|\alpha - \beta| \in O(1/D)$. Let X_i be the common point of C_i and the bisector of the walk w_i, for $i \in \{1, 2\}$, where w_1 is the walk of Blue and w_2 is the walk of Red. Then, the lemma says that, in order to make approach feasible, X_1 and X_2 should be located almost symmetrically with respect to the line tangent to C_1 and C_2. By moving one of these points on its circle by a distance $O\left(\frac{1}{D} \cdot 2\pi D\right) = O(1)$, the positions of those points would be ideally symmetric.

▶ **Lemma 7.** *Consider the set $\mathcal{I}$ of geometric instances supported by a set $\{w_A, w_B\}$ of walks. Let α be the walk angle of agent A, and β the walk angle of agent B. Then, for $D > 100$ and for any $I \in \mathcal{I}$, $\gamma(I) \in \left[\frac{\alpha+\beta}{2} - \frac{6}{D}, \frac{\alpha+\beta}{2} + \frac{6}{D}\right]$.*

Lemma 7 gives us the following corollary.

▶ **Corollary 8.** *Consider the set $\mathcal{I}$ of geometric instances supported by a set (w_A, w_B) of walks. Then all angles $\gamma(I)$, for $I \in \mathcal{I}$, belong to an interval of length at most $12/D$, for $D > 100$.*

4.3 High-level ideas of lower bounds for scenarios without simultaneous start or without common orientation

In this section we describe the ideas of the lower bounds $\Omega(D^2\sqrt{D})$ presented in Sect. 4.4 and 4.5.

4.3.1 Time-shifted instances

In order for a pair of walks $\{w_1, w_2\}$ to yield approach, both walks have to be sufficiently close geometrically (i.e., their trajectories should be close to each other). These requirements, combined with the assumption that the distance between the bases of agents is the largest still allowing approach, lead to Lemmas 6 and 7. Additionally, the walks w_1 and w_2 assuring approach should be somehow synchronized. This synchronization is necessary in order to assure that the agents get to the closest parts of the pair of walks at the same time. This property should be satisfied regardless of the delay of starting times of agents, chosen by the adversary.

Lemma 10 formalizes the above intuition and expresses some conclusions regarding time T needed for approach in the following way. First, we restrict attention to such instances I that the angle $\gamma(I)$ belongs to one of the angle-intervals from the set $\{g_i\}_{i=1}^{z}$ for $z = \Theta(\sqrt{D})$ such that:

- the length of each interval is $\Theta(1/\sqrt{D})$,
- the intervals g_i and $g_{1+i \mod z}$ are separated by an interval of length $\Theta(1/\sqrt{D})$.

Then, by Lemma 6, for all instances I, I' such that $\gamma(I) \in g_i$, $\gamma(I') \in g_{i'}$ for $i \neq i'$, there is no pair of walks which simultaneously supports I and I'. Actually, Lemma 6 implies even a stronger property that no walk w_1 belongs to the sets $\{w_1, w_2\}$, $\{w_1, w_2'\}$ such that the former set supports an instance I and the latter supports I' such that $I \in g_i$, $I' \in g_{i'}$, $i \neq i'$.

Now, assume that one agent starts at time 0 and the start of the other agent is chosen uniformly at random in the period $[0, cT]$ for some constant c, where T is the time of approach of an analyzed algorithm. For this distribution, let p_i and q_i be random variables corresponding to the fractions of time serving the interval g_i by the agents Blue and Red, respectively. Let $r_i(t)$ be the expected fraction of time in which both agents simultaneously are in their walks which could support instances from g_i in some pairs, for time shift t.

In Lemma 11, we prove a lower bound on the time which agents need to spend in parallel in their walks designated for any interval g_i of length $\Omega(1/\sqrt{D})$, in order to achieve the approach for those time-shifted instances I for which the angle $\gamma(I)$ belongs to g_i. It turns out that this time has to be $\Omega\left(D\sqrt{D}\right)$. An intuition behind this bound is that an interval of size $\Omega(1/\sqrt{D})$ requires $\Omega\left(\frac{1/\sqrt{D}}{1/D}\right) = \Omega(\sqrt{D})$ separate pairs of walks due to Lemma 7 and each change of a pair of walks takes $\Omega(D)$ time. Importantly, walks of both agents cannot help with approach for $\gamma(I) \in g_{i'}$ and $i' \neq i$, since g_i is separated from g_{i-1} and g_{i+1} by intervals of lengths $\Omega(1/\sqrt{D})$ as well – cf. Lemma 6.

In Lemmas 11 and 12 we take advantage of the fact that time shift is an appropriately chosen random variable and each walk can support instances from only one of the intervals g_i:

- In Lemma 11 we show that, for randomly chosen time-shift in $[0, 10T] \subset \mathbb{R}$ with uniform distribution, the expected value of time fraction when both agents perform walks supporting approach for the value of $\gamma(I)$ in g_i, for each i, is at least $1.1cp_iq_i$.
- Using purely algebraic arguments we show in Lemma 12 that the minimum of p_iq_i is $\Omega(1/D)$. This fact holds by substituting the maximum value of i which is $O(\sqrt{D})$ in the place of n in the lemma.

In Theorem 13 we combine the above observations (Lemmas 11 and 12) with Lemma 10, proving a lower bound on the time spent performing walks in the interval g_i, for each i and for each starting time of agent B.

4.3.2 Orientation-shifted instances

For the model without common orientation, we carefully choose the set $\mathcal{I}$ of orientation-shifted instances of size $\Theta(D\sqrt{D})$ such that no pair of walks can support more than one instance from $\mathcal{I}$. The set $\mathcal{I}$ consists of instances $I_{i,j}$ for natural i and j, such that the largest i is $O(\sqrt{D})$, the largest j is $O(D)$, $\gamma(I_{i,j})$ is i times an appropriate value of the order $\Theta(1/\sqrt{D})$ and $\phi(I_{i,j})$ is equal to j times an appropriate value of the order $\Theta(1/\sqrt{D})$. Thus,

- If $i_1 \neq i_2$ then the instances I_{i_1,j_1}, I_{i_2,j_2} cannot be supported by the same pair of walks due to the purely geometric argument regarding distance between tangency points of different geometric instances, see Lemma 6.
- If $i_1 = i_2$ and $j_1 \neq j_2$ then the instances I_{i_1,j_1}, I_{i_2,j_2} cannot be supported by the same pair of walks due to the fact that the differences between orientation shifts of those instances contradict the geometric argument regarding relative locations of walks bisectors of the walks from the set if both instances are supported by the considered pair of walks (see Lemma 7).

The above observations imply that $\Omega(D\sqrt{D})$ pairs of walks have to be scheduled in such a way that both agents performing walks of a pair have to be in the meeting parts of their walks at the same time. However, as an agent is at distance $\geq D/2$ from its base in a meeting part of its walk, there is time $\geq D$ between two consecutive pairs of walks supporting some instances from $\mathcal{I}$. This observation leads to the lower bound expressed in Theorem 15. However, some extra consideration is needed in order to tackle the case that an agent may move away by distance $> D/2$ from its base which prevents it from getting back to the base. This can happen in the final walk, at the end of which the agent does not return to the base. This special case is tackled in the final part of the proof of Theorem 15.

4.4 Time $\Omega(D^2\sqrt{D})$ for arbitrary start delay

In this section, we prove the lower bound $\Omega(D^2\sqrt{D})$ on the time of approach that holds if arbitrary start delay is permitted, even if agents have the same orientation. Some missing proofs of lemmas will be provided in the full version of the paper.

Consider an instance I of the approach problem consisting of bases a of agent A and b of agent B, and of a delay between the starting times of the agents. As I is a geometric instance with an additional delay parameter we will call it a *time-shifted instance*. We say that a pair of walks W of agents A and B supports the time-shifted instance I if W supports the geometric part of the instance I.

Let $z = 2\lfloor\sqrt{D}\rfloor$, and let $g_i = [(2i-1)\frac{2\pi}{z}, 2i\frac{2\pi}{z})$ for $i = 1, 2, \ldots, z/2$.
The following corollary is implied directly by Lemma 6.

▶ **Corollary 9.** *Consider any walk w of one of the agents, and any walks w_1, w_2 of the other agent. Let I_1 and I_2 be time-shifted instances supported by pairs of walks $\{w, w_1\}$ and $\{w, w_2\}$, respectively. Then $\gamma(I_1)$ and $\gamma(I_2)$ cannot belong to different segments g_i.*

For any $i = 1, 2, \ldots, z/2$, we define the set P_i of walks of agent A and the set Q_i of walks of agent B as follows.

We say that $w \in P_i$ (resp. $w \in Q_i$), if there exists a walk w' of agent B (resp. of agent A), such that the pair of walks $\{w, w'\}$ supports a time-shifted instance I with $\gamma(I) \in g_i$. Note that, due to Corollary 9, sets P_i, for different indices i, and sets Q_i, for different indices i, are pairwise disjoint.

Without loss of generality, we will assume that agent A starts first. Let 0 be the time when agent A starts, and $t \geq 0$ be the time when agent B starts. The local time of each agent is the time counted since its start. Assume that T is the approach time counted from the start of agent B, for a worst-case instance.

Denote by p_i the fraction of time spent by agent A in walks from set P_i in the period $[0, 11T]$. Denote by q_i the fraction of time spent by agent B in walks from set Q_i in the period $[0, T]$. (Recall that, we measure time for agent B according to its local time. In particular time 0 for agent B is equal to time t at which B starts the execution of the algorithm.) By definition, $\sum_i p_i \leq 1$ and $\sum_i q_i \leq 1$.

We define the indicator function $\mathbb{I}^i_A(s)$ as follows: $\mathbb{I}^i_A(s) = 1$, if at time s, agent A performs a walk from the set P_i and $\mathbb{I}^i_A(s) = 1$ otherwise. Analogously, $\mathbb{I}^i_B(s) = 1$ if at its local time s, agent B performs a walk from the set Q_i and $\mathbb{I}^i_B(s) = 0$ otherwise.

Note that $p_i = \frac{1}{11T}\int_0^{11T} \mathbb{I}^i_A(s)\,ds$ and $q_i = \frac{1}{T}\int_0^{T} \mathbb{I}^i_B(s)\,ds$. The *common period of walks* in P_i and Q_i is defined as the set of time points when agent A performs a walk from P_i while agent B performs a walk from Q_i. The *attempt period of walks* in P_i and Q_i is the subset of the common period of walks in P_i and Q_i consisting of time points such that both walks are in their meeting parts. This set of time points depends on the starting time of agent B. Moreover define $r_i(t) = \frac{1}{T}\int_0^{T} \mathbb{I}^i_A(t+s)\mathbb{I}^i_B(t)\,ds$. This is the fraction of the time segment $[t, t+T]$ occupied by the common period of walks in P_i and Q_i.

▶ **Lemma 10.** *Suppose that, for some $i = 1, 2, \ldots, z/2$ and some starting time t of agent B, the inequality $r_i(t) \cdot T < \left(\frac{\pi\sqrt{D}}{12} - 1\right)(D-2)$ holds. Then there exists a time-shifted instance I such that $\gamma(I) \in g_i$ and t is the starting time of agent B, for which approach does not happen.*

▶ **Lemma 11.** *Assume that agent B starts in time taken uniformly at random from the range $[0, 10T]$. The expected value of r_i is not greater than $1.1p_iq_i$.*

Proof. The expected value of r_i is

$$\begin{aligned}
\mathbb{E}[r_i] &= \frac{1}{10T}\int_0^{10T} r_i(t)\,dt = \frac{1}{10T}\int_0^{10T}\frac{1}{T}\int_0^{T} \mathbb{I}^i_A(t+s)\mathbb{I}^i_B(s)\,ds\,dt \\
&= \frac{1}{10T^2}\int_0^{T} \mathbb{I}^i_B(s)\int_0^{10T} \mathbb{I}^i_A(t+s)\,dt\,ds \\
&\leq \frac{1}{10T^2}\int_0^{T} \mathbb{I}^i_B(s)\int_0^{11T} \mathbb{I}^i_A(t)\,dt\,ds \\
&\leq \frac{11}{10}\frac{1}{11T}\int_0^{11T} \mathbb{I}^i_A(t)\,dt \cdot \frac{1}{T}\int_0^{T} \mathbb{I}^i_B(s)\,ds = 1.1p_iq_i
\end{aligned}$$

◀

To complete our considerations, we will also need the following algebraic lemma.

▶ **Lemma 12.** *Consider real numbers $p_i, q_i \in [0, 1]$, for $i = 1, \ldots, n$, such that $\sum_{i=1}^n p_i \leq 1$ and $\sum_{i=1}^n q_i \leq 1$. Then $\min\{p_iq_i : i \leq n\} \leq \frac{1}{n^2}$.*

▶ **Theorem 13.** *The time of approach for two agents with arbitrary start delay, starting at a distance at most D, is $\Omega(D^2\sqrt{D})$.*

Proof. We apply Lemma 12 for $n = z/2 = \lfloor\sqrt{D}\rfloor$ and for p_i, q_i defined before in this section. By Lemma 12, there exists an i for which $p_iq_i \leq 1/n^2 \leq 4/D$. By Lemma 11, the expected value of r_i is at most $1.1p_iq_i$. Thus there exists an argument t for which $r_i(t) \leq 1.1p_iq_i$. Suppose that the approach occurs for an instance I for which $\gamma(I) \in g_i$ and the delay between starting times of the agents is t. By Lemma 10, we have $r_i(t) \cdot T \geq \left(\frac{\pi\sqrt{D}}{12} - 1\right)(D-2)$, where T is the length of time between the start of agent B and the approach. Hence

$$\left(\frac{\pi\sqrt{D}}{12} - 1\right)(D-2) \leq r_i(t)T \leq 1.1p_iq_iT \leq 5T/D.$$

Thus we get $T \geq \frac{D}{5}\left(\frac{\pi\sqrt{D}}{12} - 1\right)(D-2) \in \Omega(D^2\sqrt{D})$ which proves the theorem. ◀

4.5 Time $\Omega(D^2\sqrt{D})$ for arbitrary orientations

In this section, we prove the lower bound $\Omega(D^2\sqrt{D})$ on the time of approach that holds if orientations of agents may be different and arbitrary, even if they start simultaneously.

We will call the orientation of agent A the *global orientation*. Consider an instance I of the approach problem consisting of bases a of agent A and b of agent B, and of the orientation of agent B. As it is a geometric instance with an additional rotation parameter we will call it an *orientation-shifted instance*. Let $\phi(I)$ be a the angle between the half-line indicating north according to the local orientation of agent B and the halfline indicating north in the global orientation. Thus, an orientation-shifted instance I is determined by the locations a and b of the bases of agents and the angle $\phi(I)$. Let $\{w_A, w_B\}$ be a pair of walks, where w_A, w_B are walks of agent A and agent B, respectively. Let $w_B(I)$ be the walk w_B of agent B rotated around the base of agent B by $\phi(I)$, i.e., the walk w_B expressed according to the global orientation in the orientation-shifted instance I. We say that the pair of walks $\{w_A, w_B\}$ *supports* the orientation-shifted instance I if $\{w_A, w_B(I)\}$ supports the part of the instance I determined by the locations of the bases.

Let $\mathcal{I}$ be a set of orientation-shifted instances $I_{i,j}$ for $i \in \{0, \ldots, \lfloor 2\pi\frac{\sqrt{D}}{6}\rfloor - 1\}$ and $j \in \{0, \ldots, \lfloor 2\pi\frac{D}{30}\rfloor - 1\}$ such that:

- the distance between the bases of agents A and B is exactly D.
- $\gamma(I_{i,j}) = \frac{6i}{\sqrt{D}}$,
- $\phi(I_{i,j}) = \frac{30j}{D}$.

▶ **Lemma 14.** *Let $\mathcal{S}$ be a collection of pairs of walks. If each instance $I \in \mathcal{I}$ is supported by a pair of walks from $\mathcal{S}$ then $|\mathcal{S}| \geq |\mathcal{I}|$.*

Proof. It is sufficient to prove that each pair of walks can support only one instance from the set $\mathcal{I}$. For the sake of contradiction assume that there exists a pair of walks $S \in \mathcal{S}$ supporting two different instances $I_{i_1,j_1}, I_{i_2,j_2} \in \mathcal{I}$ for $i_1, i_2 \in \{\{0, \ldots, \lfloor 2\pi\frac{\sqrt{D}}{6}\rfloor - 1\}\}$ and $j_1, j_2 \in \{0, \ldots, \lfloor 2\pi\frac{D}{30}\rfloor - 1\}$. Let us analyze two cases.

- $i_1 \neq i_2$. Let α be the walk bisector of the walk of agent A. By Lemma 6, we need both inequalities $|\alpha - \gamma(I_{i_1,j_1})| \leq 3/(2\sqrt{D})$ and $|\alpha - \gamma(I_{i_2,j_2})| \leq 3/(2\sqrt{D})$ in order to assure that S supports I_{i_1,j_1} and I_{i_2,j_2}. However, these inequalities imply that $|\gamma(I_{i_1,j_1}) - \gamma(I_{i_2,j_2})| \leq 3/\sqrt{D}$ and, by the definition of $\mathcal{I}$, $|\gamma(I_{i_1,j_1}) - \gamma(I_{i_2,j_2})| \geq 6/\sqrt{D}$ which gives a contradiction.
- $i_1 = i_2$ and $j_1 \neq j_2$.
  Let α be the walk bisector of the walk of agent A and let $\beta(I_{i_k,j_k})$ be the walk bisector of the walk of agent B with respect to the global orientation for the instance I_{i_k,j_k} where $k \in \{1, 2\}$. By Lemma 7, we need that both $\gamma(I_{i_1,j_1}) \in \left[\frac{\alpha+\beta(I_{i_1,j_1})}{2} - \frac{6}{D}, \frac{\alpha+\beta(I_{i_1,j_1})}{2} + \frac{6}{D}\right]$ and $\gamma(I_{i_2,j_2}) \in \left[\frac{\alpha+\beta(I_{i_2,j_2})}{2} - \frac{6}{D}, \frac{\alpha+\beta(I_{i_2,j_2})}{2} + \frac{6}{D}\right]$ hold, in order to assure that S supports both I_{i_1,j_1} and I_{i_2,j_2}.
  However, as $\gamma(I_{i_1,j_1}) = \gamma(I_{i_2,j_2})$, $\beta(I_{i_1,j_1}) = \beta(I_{i_2,j_2}) + \phi(I_{i_1,j_1}) - \phi(I_{i_2,j_2})$, and, given that $|\phi(I_{i_1,j_1}) - \phi(I_{i_2,j_2})| \geq \frac{30}{D}$, the segments

$$\left[\frac{\alpha + \beta(I_{i_1,j_1})}{2} - \frac{6}{D}, \frac{\alpha + \beta(I_{i_1,j_1})}{2} + \frac{6}{D}\right]$$

and

$$\left[\frac{\alpha+\beta(I_{i_2,j_2})}{2}-\frac{6}{D},\frac{\alpha+\beta(I_{i_2,j_2})}{2}+\frac{6}{D}\right]$$
$$=\left[\frac{\alpha+\beta(I_{i_1,j_1})}{2}+\frac{\phi(I_{i_1,j_1})-\phi(I_{i_2,j_2})}{2}-\frac{6}{D}\right.,$$
$$\left.\frac{\alpha+\beta(I_{i_1,j_1})}{2}+\frac{\phi(I_{i_1,j_1})-\phi(I_{i_2,j_2})}{2}+\frac{6}{D}\right]$$

are disjoint, which gives a contradiction. ◀

Using Lemma 14, we can prove the main theorem of this section.

▶ **Theorem 15.** *The time of approach for two agents with arbitrary orientations, starting at a distance at most D is $\Omega(D^2\sqrt{D})$, even if agents start simultaneously.*

Proof. The proof is by contradiction. Assume that there is an approach algorithm for two agents with arbitrary orientations, starting at a distance D, which completes the approach task in time at most $\frac{1}{32}\left(\frac{\sqrt{D}}{6}\cdot\frac{D}{30}-1\right)(D-2)+D$ for all instances in $\mathcal{I}$.

First, consider only walks wholly contained inside the range circles of radius $D/2$ of the respective agents. We say that a pair of walks $\{w_1,w_2\}$ is *useful* if there exists a time t such that both w_1 and w_2 are in their meeting areas at time t.[2] The pair of walks $\{w_1,w_2\}$ cannot lead to approach in any instance, if it is not useful. (The reason is that, in such a case, at each time t the actual distance between the agents is larger than 1, regardless of the current instance.) A *usefulness witness* of a useful set $\{w_1,w_2\}$ is the smallest time t such that both walks are in their meeting areas at time t. Consider the sequence $t_1<t_2<\cdots<t_p$, for some positive integer p, which consists of all usefulness witnesses of all pairs of walks (one walk for agent A, and one for agent B) of the algorithm. Then, by Lemma 14, at most p instances from $\mathcal{I}$ could potentially be supported by the algorithm. On the other hand, $t_l-t_{l-1}\geq D-2$ for each $l\in[2,p]$ because at least one agent has to move back to its base from a point at distance $\geq D/2-1$, and then move away from the base at a distance $\geq D/2-1$, in the time period $[t_{l-1},t_l]$. Therefore, p is at most $\frac{1}{32}\cdot\left(\frac{\sqrt{D}}{6}\frac{D}{30}-1\right)$ which is smaller than $\frac{1}{32}$th of the size of $\mathcal{I}$ – a contradiction.

Thus, for at least $\frac{31}{32}|\mathcal{I}|$ instances in $\mathcal{I}$ the approach must occur when at least one agent leaves its range disc[3]. Denote the set of such instances by $\mathcal{I}'$.

Recall that, by the definition of a walk, the agent gets back to its base at the end of each walk, except the final walk that ends with approach. As an agent is not able to get back to the base after leaving its range disc, the only walk of the agent after it leaves the range disc can be the final walk. Let's call the walk during which an agent leaves its range disc a *special walk*.

Hence the approach for all instances from the set $\mathcal{I}'$ (whose size is $\geq\frac{31}{32}|\mathcal{I}|$) must occur during the special walk of at least one agent. (However, many other walks of the other agent might occur during this special walk.)

Let $n=\lfloor 2\pi\frac{\sqrt{D}}{6}\rfloor$ and $m=\lfloor 2\pi\frac{D}{30}\rfloor$. For clarity of presentation we assume that both n and m are divisible by 4. One can easily verify that the proof holds in the general case as well, i.e., when n or m is not necessarily divisible by 4.

[2] Note that we work in the model with common start here, so both agents work according to the same time.

[3] Recall that the range disc of an agent is the disk with radius $D/2$ centered at the base of the agent.

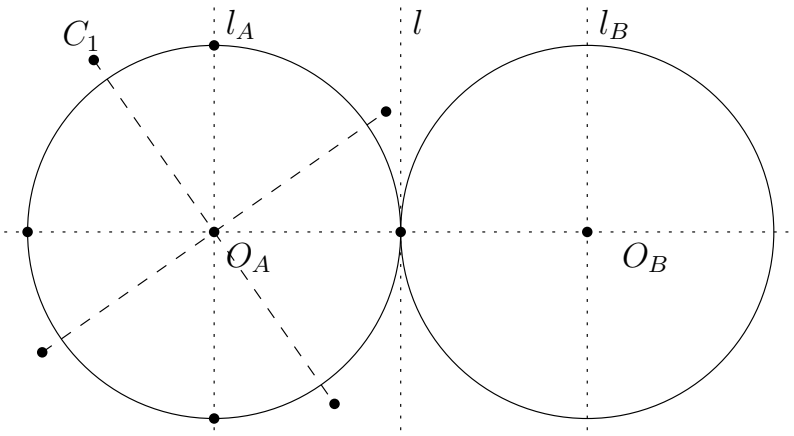


Figure 4 The location of the point C_1 in the instance $I'_{a,b}$. The dots at the ends of the dashed segments illustrate all possible locations of C_1 for instances $I'_{a,b}$ for arbitrary $b \in [0,3]$ and various values of $a \in [0,3]$.

Now, let

$$\mathcal{I}_{i,j} = \{I'_{a,b} \mid I'_{a,b} = I_{i+a\cdot\frac{n}{4},j+b\cdot\frac{m}{4}} \text{ for each } a,b \in [0,3]\}$$

for $i \in [0, n/4-1]$ and $j \in [0, m/4-1]$. Since $|\mathcal{I}'| \geq \frac{31}{32}|\mathcal{I}|$ and $\mathcal{I}$ can be split into $|\mathcal{I}|/16$ subsets $\mathcal{I}_{i,j}$, there exists a subset $\mathcal{I}_{i,j}$ for some $i \in [0, n/4-1]$ $j \in [0, m/4-1]$ which is entirely included in $\mathcal{I}'$.

Thus, let $i \in [0, n/4-1]$ $j \in [0, m/4-1]$ be such that $\mathcal{I}_{i,j} \subset \mathcal{I}'$. Hence, none of the pairs of non-special walks of the agents are such that an approach occurs during executions of some of those pairs of walks for any of the 16 instances from $\mathcal{I}_{i,j}$.

We now show that the other agent must also leave its range disc in order to ensure approach for all instances from $\mathcal{I}_{i,j}$. The proof is by contradiction. Assume that it is not the case. Without loss of generality assume that A is the only agent which leaves its range disc. Let C_1 be a point in the only special walk of A, at distance $D/2+\varepsilon_1$ from its base for some constant $\varepsilon_1 > 0$, such that A is at C_1 before approach happens for any of the instances from $\mathcal{I}_{i,j}$ (recall that approach cannot happen as long as both agents are inside their range discs, according to the definition of $\mathcal{I}_{i,j}$). Observe that, by choosing different values of $a \in [0,3]$, we can change the location of C_1 with respect to the base O_B of B in the instances $I'_{a,j}$ for all $j \in [0,3]$. Indeed, as we assume that the orientation of A corresponds to the global orientation, the value of $\gamma(I'_{a,b})$ determines relative location of C_1 and O_B as illustrated on Figure 4. The value of $\gamma(I'_{a,b})$ is determined by the value of a. Let $a \in [0,3]$ be such that the base O_B of B is furthest from the point C_1 in $I'_{a,b'}$, for $b' \in [0,3]$. Let l be the line tangent to the range circles of both agents, let l_A and l_B be the lines parallel to l, going through the bases O_A and O_B of the agents, respectively – see Figure 4. Then, in the instance $I'_{a,b}$ for $b \in [0,3]$, the point C_1 is on the opposite side of l_A than the tangency point of the range circles of the agents. And therefore C_1 is further from the tangency point of the range circles of the agents than from the base of the agent A and so it is further from the base of B than $D+1$. Indeed, as $|C_1O_A| > D/2$, $|O_AO_B| = D$ and the angle $\angle C_1O_AO_B$ is obtuse, the Pythagorean Theorem implies that $|C_1O_B| > \sqrt{|C_1O_A|^2 + |O_AO_B|^2} > \sqrt{(D/2)^2 + D^2} > D+1$ for $D > 5$. As A has at this point only $D/2-\varepsilon_1$ fuel left, the agent B must in this case travel further from its base than $D/2$ in order to ensure approach of the agents. But this fact contradicts our assumption that only one of the agents leaves its range disk during the execution of the algorithm.

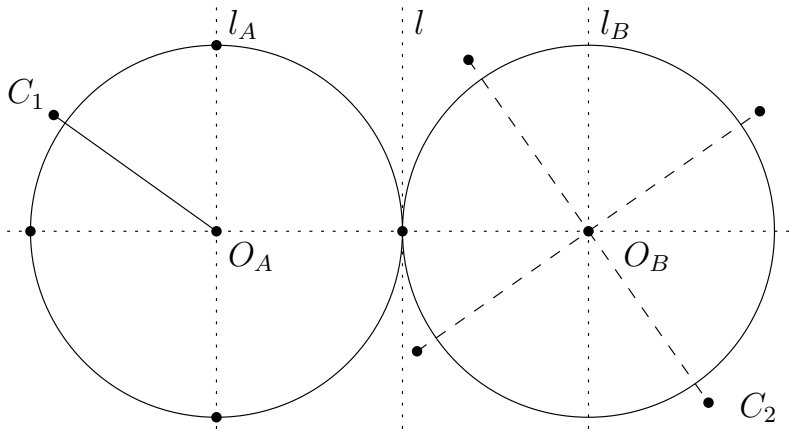


Figure 5 The locations of the points C_1 and C_2 in the instance $I'_{a,b}$. The dots at the ends of the dashed segments illustrate all possible locations of C_2 for instances $\mathcal{I}_{a,b'}$, $b' \in [0,3]$.

Let C_2 be any point in the special walk of B at distance $D/2 + \varepsilon_2 > 0$ from the base of agent B. Again, let b be such that the greatest distance between C_1 and C_2 amongst all the instances $I'_{a,b'}$ for $b' \in [0,3]$ is obtained for $I'_{a,b}$. Then, the point C_2 is located on the opposite side of l_B than C_1 in $I'_{a,b}$ – see Figure 5. Inspecting the triangle $\Delta C_1 O_A C_2$ with the obtuse angle $\angle C_1 O_A C_2$, we obtain the following lower bound on the distance between C_1 and C_2: $|C_1C_2| > \sqrt{|C_1O_A|^2 + |O_AC_2|^2} > \sqrt{(D/2)^2 + D^2} \geq D + 1$ for $D > 5$, by the Pythagorean Theorem. On the other hand, the amount of fuel left in the tank of A at the point C_1 and the amount of fuel left in the tank of B at the point C_2 are smaller than $D/2$ since the agents are outside of their range discs at these points. Thus, the approach cannot occur for the instance $I'_{a,b}$.

As our assumption led to an occurrence of an instance without approach, the algorithm working in time $\frac{1}{32} \cdot \left(\frac{\sqrt{D}}{6}\frac{D}{30} - 1\right)(D-2) + D$ cannot ensure the approach for all instances from $\mathcal{I}$, which concludes the proof . ◀

5 Discussion

In this section, we discuss two assumptions of our model. The first is that the size of the tank (i.e., the distance that an agent can travel without replenishing the tank) is D, and the second is that the time of approach is counted from the wake-up of the later agent.

Considering the first assumption, it is natural to ask if a smaller tank wouldn't be enough. First observe that an agent with tank of size x cannot go at distance larger than $x/2$ from its base, possibly apart from the last trip that does not end at the base, because then it would be unable to get back to the base in order to replenish its tank. This implies that the size of the tank must be strictly larger than $D-1$. Indeed, consider two agents at distance exactly D. If the size x of the tank were strictly smaller than $D-1$ then agents could never approach without going farther than $x/2$ from their bases, as their distance would be always larger than 1. It can also be shown that the adversary can place the bases of agents in such a way that approach could not happen during the final trip, when an agent does not have to get back to the base.

If the size of the tank were exactly $D-1$ then the only way to approach without travelling farther than $(D-1)/2$ from the base would be for both agents to go straight on the segment joining their bases, each at distance exactly $(D-1)/2$. Clearly, for any hypothetical algorithm

of approach, the adversary can place the bases in order to avoid such a perfect guess of direction in any finite time. Similarly as before, the adversary can prevent approach during the final trip.

The remaining case is that of a tank of size $(D-1)+\epsilon$, for some $0<\epsilon<1$. By slightly modifying our algorithms, it can be shown that, for each of our models, approach can be guaranteed with the same time complexity, whenever the tanks of the agents have size $(D-1)+\epsilon$, for any constant $0<\epsilon<1$.

The second assumption is that the time of approach is counted from the wake-up of the later agent. An alternative way would be to count time from the wake-up of the earlier agent. However, for any $D>2$, consider two agents with tanks of size D and bases at distance D. Before the wake-up of the later agent, the earlier agent cannot approach it without travelling farther than $D/2$ from its base. Since the adversary can make the delay between the wake-up times of the agents arbitrarily long, the time of approach counted from the wake-up of the earlier agent would be unbounded.

6 Conclusion

We considered the problem of approach of agents with restricted tanks and designed algorithms of optimal complexity, for any of the four scenarios yielded by the presence or absence of our two coherence assumptions. In this paper, we heavily relied on the synchronous character of the agents: they can stay put for a chosen amount of time and when they travel, they always move at speed 1. It is natural to ask how the problem of approach for agents with restricted tanks changes in the asynchronous setting, where agents can choose the direction and distance of their moves but the adversary decides the speeds of the agents. In this model, agents cannot choose to wait, as the adversary controls waiting times as well. The problem of asynchronous approach for agents with unrestricted tanks has been solved in [4]. What is the solution of it for agents with restricted tanks and how large tanks are needed?

References

1 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, 2003.

2 Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Inf. Comput.*, 152(2):155–172, 1999. doi:10.1006/INCO.1999.2795.

3 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993. doi:10.1006/INCO.1993.1054.

4 Sébastien Bouchard, Marjorie Bournat, Yoann Dieudonné, Swan Dubois, and Franck Petit. Asynchronous approach in the plane: a deterministic polynomial algorithm. *Distributed Comput.*, 32(4):317–337, 2019. doi:10.1007/S00446-018-0338-2.

5 Sébastien Bouchard, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Almost-optimal deterministic treasure hunt in unweighted graphs. *ACM Trans. Algorithms*, 19(3):22:1–22:32, 2023. doi:10.1145/3588437.

6 Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012. doi:10.1137/100796534.

7 Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM J. Comput.*, 34(6):1516–1528, 2005. doi:10.1137/S0097539704446475.

8 Jurek Czyzowicz, Leszek Gasieniec, Ryan Killick, and Evangelos Kranakis. Symmetry breaking in the plane: Rendezvous by robots with unknown attributes. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 4–13. ACM, 2019. doi:10.1145/3293611.3331608.

9 Yoann Dieudonné, Andrzej Pelc, and Franck Petit. Almost universal anonymous rendezvous in the plane. *Algorithmica*, 85(10):3110–3143, 2023. doi:10.1007/S00453-023-01122-2.

10 Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015. doi:10.1137/130931990.

11 Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006. doi:10.1145/1159892.1159897.

12 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. doi:10.1016/J.TCS.2005.01.001.

13 Shmuel Gal. *Search Games: A Review*, pages 3–15. Springer New York, New York, NY, 2013. doi:10.1007/978-1-4614-6825-7_1.

14 Subir Kumar Ghosh and Rolf Klein. Online algorithms for searching and exploration in the plane. *Comput. Sci. Rev.*, 4(4):189–201, 2010. doi:10.1016/J.COSREV.2010.05.001.

15 Andrzej Pelc. Reaching a target in the plane with no information. *Inf. Process. Lett.*, 140:13–17, 2018. doi:10.1016/J.IPL.2018.04.006.

16 Andrzej Pelc. Deterministic rendezvous algorithms. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 423–454. Springer, 2019. doi:10.1007/978-3-030-11072-7_17.

17 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.

Appendix

A Proof of Lemma 1

Proof. Without loss of generality, let P_1 be counterclockwise from S. Let l be the line

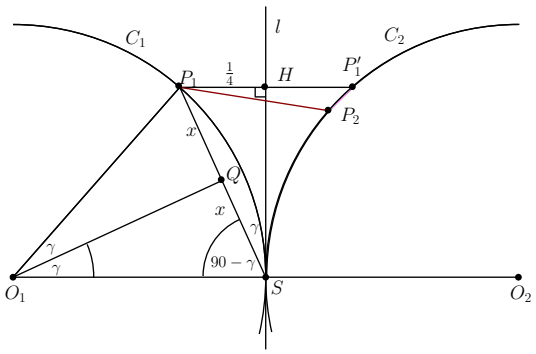


Figure 6 Illustration for Lemma 1.

tangent to C_1 in point S and let O_1, O_2 be the centres of C_1 and C_2, respectively. Let H be the common point of l and of the line parallel to the radius O_1S, passing through the point P_1. Let Q be the common point of the bisector of the angle SO_1P_1 and of the segment

P_1S. As the triangle SO_1P_1 is an isosceles triangle, O_1QP_1 is a right triangle, and thus the triangles O_1QP_1 and SHP_1 are similar, because $|\angle QO_1P_1| = |\angle HSP_1|$ (cf. Fig. 6).

As $\alpha_1(P_1) \le y = \frac{1}{4}\sqrt{r}$, the length x of the segment P_1Q satisfies $x = \frac{1}{2}|P_1S| \le \alpha_1(P_1) \le \frac{1}{4}\sqrt{r}$.

By similarity of the triangles O_1QP_1 and SHP_1, we have $\frac{|O_1P_1|}{|P_1Q|} = \frac{|P_1S|}{|P_1H|}$ and thus

$$|P_1H| = \frac{|P_1Q| \cdot |P_1S|}{|O_1P_1|} = 2\frac{x^2}{r} \le 1/8. \tag{1}$$

Let P_1' be the point axial symmetric to P_1 with respect to axis l. As l is also tangent to circle C_2, and C_2 has the same radius as C_1, the point P_1' lies on C_2.

In view of the assumption $\alpha_1(P_1) - 1/2 \le \alpha_2(P_2) \le \alpha_1(P_1) + 1/2$ we have

$$|P_1'P_2| \le |\alpha_2(P_1') - \alpha_2(P_2)| = |\alpha_1(P_1) - \alpha_2(P_2)| \le \frac{1}{2}. \tag{2}$$

By the triangle inequality, (1), and (2) we have $|P_1P_2| \le |P_1P_1'| + |P_1'P_2| = 2|P_1H| + |P_1'P_2| \le \frac{1}{4} + \frac{1}{2} < 1$. ◀

B Proof of Theorem 2

Proof. Suppose that the distance between the base O_1 of agent Blue and the base O_2 of agent Red is $d \le D$. Let C_1 and C_2 be circles of radius $D/2$ centered at O_1 and O_2 respectively, and let C_1' and C_2' be circles of radius $d/2$ centered at O_1 and O_2 respectively. Let S be the unique common point of circles C_1' and C_2'.

Consider points $A_0, \ldots, A_{n-1}$ on circle C_1, visited by agent Blue in its consecutive phases. For $i = 0, 1, \ldots n-1$, let A_i' be the intersection point of the segment O_1A_i with the circle C_1'. Let j be the index for which point A_j' is closest to point S among all points A_i'. If there are two such closest points, j is chosen arbitrarily between their indices.

Consider the time point t in phase j when agent Blue is at point A_j' for the first time. At time t, agent Red is at some point B on circle C_2'. We use Lemma 1 substituting $d/2$ for r, B for P_1, and A_j' for P_2. Denote by α the length of the arc SB on circle C_2'. In view of the definition of n, we have $\alpha \le \frac{1}{4}\sqrt{d/2}$ and the length of arc SA_j' on circle C_1' is between $\alpha - 1/2$ and $\alpha + 1/2$. Thus both assumptions of Lemma 1 are satisfied. This implies $|A_j'B| < 1$. Hence approach occurs in phase j. Since $j < n = 4\lceil \pi D \rceil$ and the duration of each phase is D, phase j is completed in time $O(D^2)$ which proves the theorem. ◀

C Proof of Theorem 3

Proof. Suppose that the distance between the base O_1 of agent Blue and the base O_2 of agent Red is $d \le D$. Consider two cases.

Case 1. $d = D$.

In this case, the circles C_1 and C_2 are tangent. Let S be the point of tangency. Let $j \in [0, n-1]$ be such an index that the point A_j is closest to the point S among all points A_i. If there are two such closest points, j is chosen arbitrarily from the indices of these two closest points. Let k be the index for which B_k is the closest point from the set $\{B_0, \ldots, B_{m-1}\}$ to the point A_j. Consider the time point t when agent Blue visits the point A_j for the first time after both agents have been awaken. As agent Blue waits in this point for time $4\lceil 4\pi D^2 \rceil$, there is a time point $t' \in [t, t + 4\lceil 4\pi D^2 \rceil]$, when agent Red visits the point B_k. Now, we may use Lemma 1 substituting $d/2 = D/2$ for r, A_j for P_1, and B_k for P_2. The definition of n

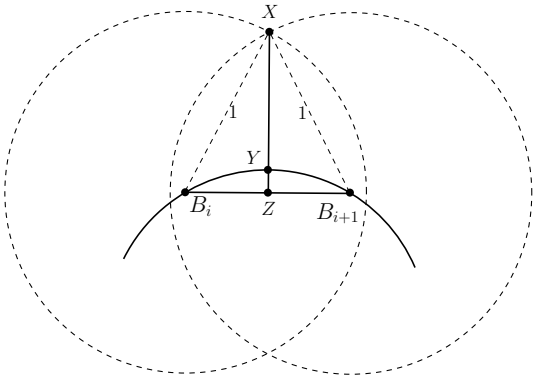


Figure 7 Illustration for Claim 16.

implies that the length α of arc SA_j on the circle C_1 is at most $\frac{1}{4}\sqrt{D/2}$ and the length of arc SB_k on the circle C_2 is not smaller than $\alpha - 1/2$ and not larger than $\alpha + 1/2$. Thus both assumptions of Lemma 1 are satisfied. This implies the inequality $|A_jB_k| < 1$. Hence approach occurs at time t'. Since t is the first time agent Blue visits point A_j when both agents are awake, the approach occurs in time at most $4\lceil 4\pi D^2\rceil(\lceil 2\pi\sqrt{D}\rceil + 2) = O\left(D^2\sqrt{D}\right)$.

Case 2. $d < D$.

Let the *approach area* of an agent in some time period be the area equal to all points at distance ≤ 1 from all points visited by the agent during the given time period. The following claim provides significant property of the approach area of agent Red during its execution of the algorithm. Then, in Claim 17, we will show that the property stated in Claim 16 guarantees that A_j belongs to the approach area of agent Red for some $j \in [0, n-1]$.

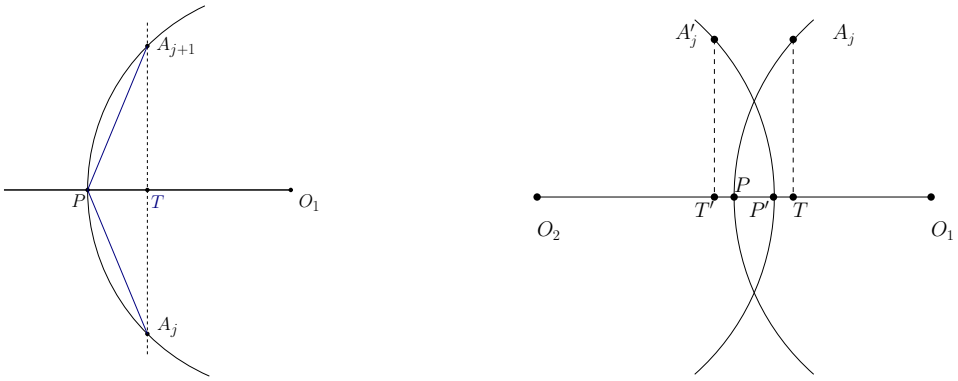
▷ Claim 16. Each point within distance of $\frac{1}{2}D + \frac{\sqrt{15}-1}{4}$ from the base of agent Red belongs to its approach area.

In order to prove the claim, notice that, as the length of the circle C_2 is $2\pi \cdot \frac{1}{2}D = \pi D$, the distance between B_i and $B_{(i+1) \mod m}$ is smaller than one, and therefore the set of points at distance at most 1 from the set of points $\{B_0, \ldots, B_{m-1}\}$ covers the whole circle C_2. As agent Red moves via a straight line from its base to each of the points B_i, its approach area covers C_2. Now we will show that agent Red also approaches the additional annulus of width at least $\frac{\sqrt{15}-1}{4}$ outside of C_2.

Fix two consecutive points $B_i, B_{(i+1) \mod m}$ on C_2. (For brevity, we will write B_{i+1} instead of $B_{(i+1) \mod m}$ in the rest of the proof.) Consider the circles of radius one with the centres in B_i and B_{i+1}, respectively (see Fig. 7). The union of the corresponding discs is the area approachable from both B_i and B_{i+1}. Let X be a common point of these two discs at the largest distance from O_2. The point X is one of the points furthest from O_2 at distance 1 from B_i. Our goal is to bound the distance from X to O_2.

Consider the segment going through the point X and perpendicular to B_iB_{i+1}. Let Y, Z be, respectively. the common point of this segment with the circle C_2 and with the segment B_iB_{i+1} (see Fig. 7).

As X is a common point of circles of the same radius, the point Z divides the segment of B_iB_{i+1} into two segments of equal lengths. Similarly, the point Y divides the arc B_iB_{i+1} in two arcs of equal lengths. As there are $m = \lceil 4\pi D\rceil$ points B_j, the length of B_iB_{i+1} is at most $1/4$ and, as B_iX is a radius of length 1, we have $|XZ| = \sqrt{1 - |B_iZ|^2} \geq \sqrt{1 - \frac{1}{16}} = \frac{\sqrt{15}}{4}$.



(a) Locations of A_j, A_{j+1}, P and T.

(b) Locations of A'_j, P' and T'.

Figure 8 Illustration of notations from the proof of Claim 17.

Moreover, the arc B_iY has length $|\widehat{B_iY}| \leq \frac{|B_i\widehat{B}_{i+1}|}{2} \leq \frac{1}{2} \cdot \frac{2\pi \cdot \frac{1}{2}D}{|4\pi D|} < \frac{1}{4}$. As the segment B_iY is a hypotenuse of the triangle B_iZY we have $|YZ| \leq |B_iY| \leq \frac{1}{4}$ and $|XY| = |XZ| - |YZ| \geq \frac{\sqrt{15}}{4} - 1/4 = \frac{\sqrt{15}-1}{4}$.

Thus the point X is at distance at least $\frac{1}{2}D + \frac{\sqrt{15}-1}{4}$ from O_1, which implies that all points at this distance are approachable by agent Red. This proves the claim. ⋄

Now we will show that there is at least one point A_i, which agent Red can approach.

▷ Claim 17. There is at least one point A_i, for $i \in [0, n-1]$, with distance at most $\frac{1}{2}D + \frac{\sqrt{15}-1}{4}$ from O_2.

In order to prove the claim notice that, if there is a point A_i, for $i \in [0, n-1]$, located inside the disc limited by circle C_2, then the claim holds. So, assume that there is no such point A_i.

Let $j \in [0, n-1]$ be an index for which A_j is the closest point to O_2 among all the points from $\{A_0, \ldots, A_{n-1}\}$. If there are two such indices, let j be such that $|O_2A_j| = |O_2A_{j+1}|$ and O_2A_j is the smallest distance from O_2 to all A_is. Let P be the common point of the circle C_1 and the line O_1O_2 such that P is located on the segment O_1O_2 (cf. Fig 8a). Moreover, let T be the projection of the point A_j on the line O_1O_2. We want to bound the distance from P to A_j from above. Observe that, the distance between A_j and P is maximized in the instance in which there are two points A_j and A_{j+1} closest to O_2 symmetric with respect to the line O_1O_2, as presented in Fig. 8a. Therefore, in order to bound the length A_jP from above, we will restrict our consideration to this specific case.

Let l be the line perpendicular to O_1O_2 going through A_j and A_{j+1}. Let T be the common point of the line O_1O_2 and the line l (see Fig. 8a).

As A_j and A_{j+1} are two consecutive points from the cyclic sequence $A_0, \ldots, A_{n-1}$, the length of the arc A_jA_{j+1} is at most $\sqrt{D}$. Thus the angle $\angle A_jPA_{j+1}$ is at least $\pi \cdot \frac{2\pi D - \sqrt{D}}{2\pi D} = \pi - \frac{1}{2\sqrt{D}}$.

As the line PT is perpendicular to the segment A_jA_{j+1}, we have $\alpha = \angle A_jPT = \frac{1}{2}\angle A_jPA_{j+1} = \frac{\pi}{2} - \frac{1}{4\sqrt{D}}$ One can verify that, for all $x \geq 1$, $\tan\left(\frac{\pi}{2} - \frac{1}{4x}\right) > 3x$. Thus

$$|PT| = \frac{|A_iT|}{\tan\alpha} < \frac{\frac{\sqrt{D}}{2}}{3\sqrt{D}} = \frac{1}{6}. \tag{3}$$

Let P' be a common point of the line O_1O_2 and the circle C_2. Moreover, let T' be the point on O_1O_2 such that $|P'T'| = |PT|$ and $|O_2T'| \leq D$. Let A'_jT' be the segment symmetric to A_jT with respect to the bisector of the segment O_1O_2. As the circles C_1 and C_2 have the same radius, the point A'_j lies on circle C_2 (see Fig. 8b).

The length of $A_jA'_j$ satisfies the inequality (see Fig. 8b) $|A_jA'_j| < |PT|+|P'T'| < \frac{1}{3}$, where the last inequlity follows from (3). Thus by the triangle inequality, $|O_2A_j| \leq |OA'_j|+|A'_jA_j| < D + \frac{1}{3} < D + \frac{\sqrt{15}-1}{4}$. This completes the proof of the Claim. $\diamond$

Let A_j be a point whose existence is guaranteed by Claim 17. Let t be the time point when agent Blue visits point A_j for the first time after both agents have been awaken. As agent Blue waits in this point for time $4\lceil 4\pi D^2\rceil$, agent Red visits all of his points B_i during this time and, by Claims 16 and 17, the approach is successful. Thus the time to complete approach is at most $4\lceil 4\pi D^2\rceil(\lceil 2\pi\sqrt{D}\rceil + 2)$. This implies that algorithm `Slower` guarantees approach in time $O(D^2\sqrt{D})$. ◀

D Proof of Theorem 5

Proof. Consider an arbitrary approach algorithm for two agents A and B. Let $a(t)$ (resp. $b(t)$) be a vector describing the position of agent A (resp. B) relative to its starting point in time t. Let $d(t) = a(t) - b(t)$. Let T be the approach time of the algorithm.

For $i, j \in [-\lfloor \frac{D}{4}\rfloor, \lfloor \frac{D}{4}\rfloor - 2]$ let $r_{i,j}$ be the rectangle with vertices in points $(2i, 2j), (2i + 2, 2j), (2i, 2j + 2), (2i + 2, 2j + 2)$.

We will consider only *even* rectangles, that is the rectangles $r_{i,j}$ where both i and j are even. We say that the algorithm *marks the rectangle* $r_{i,j}$ if there is a moment t of the execution of the algorithm such that the point $d(t)$ is inside $r_{i,j}$. As for any t_1, t_2 such that, $|t_1 - t_2| \leq 1$ the length of the vector $d(t_1) - d(t_2)$ is at most 2, the algorithm can mark at most $T + 1$ even rectangles.

Assume that the time of approach T is less than $\frac{D^2}{16}$. The algorithm can mark at most $\frac{D^2}{16} + 1$ even rectangles, and thus there exists an unmarked even rectangle $r_{i,j}$. By fixing the base of agent A at point $(0, 0)$ and the base of B at point $(2i + 1, 2j + 1)$ the adversary can guarantee that the approach does not occur. This contradiction concludes the proof. ◀

E Proof of Lemma 6

Proof. Let S be the tangent point of range circles of the agents. Let l be the line tangent to the range circle C of agent A at point S. Let P be the point of the meeting area of the walk w_A, closest to line l (see Fig. 9). Without loss of generality, suppose that α is the walk angle of agent A. Denote by σ the angle SaP. Thus, $\sigma = |\alpha - \gamma| - \delta$.

Let H be the intersection of l with the line containing P, parallel to the segment aS. Let Q be the midpoint of segment PS. Due to similarity of triangles PHS and PQa we have

$$\frac{|PH|}{|PQ|} = \frac{|PS|}{|Pa|} = \frac{2|PQ|}{D/2}$$

and thus $|PH|D = 4|PQ|^2$. Hence $|PH| > 1$ if and only if $|PQ| > \sqrt{D}/2$. Suppose that $D > 256$.

If $\sigma < 2$, then

$$|PQ| = \frac{D}{2}\sin\left(\frac{\sigma}{2}\right) \geq \frac{D}{2}\cdot 0.8\frac{\sigma}{2} > 0.4D\left(\frac{|\alpha-\gamma|}{2} - \frac{\delta}{2}\right) \geq 0.4D\left(\frac{1.5}{\sqrt{D}} - 4/D\right) > \frac{\sqrt{D}}{2}.$$

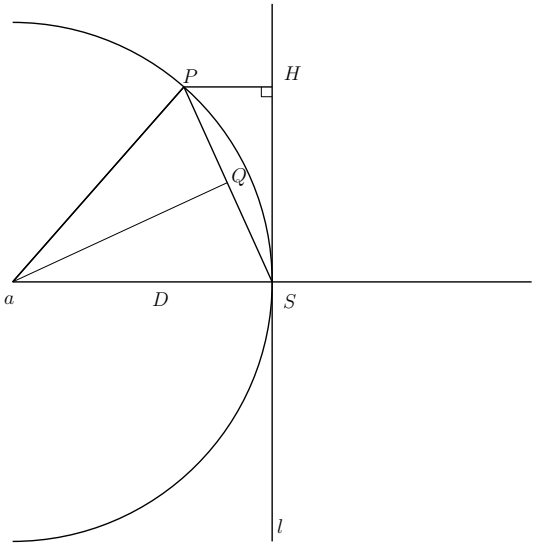


Figure 9 Illustration for Lemma 6.

If $\sigma \geq 2$, then

$$|PQ| = \frac{D}{2} \sin\left(\frac{\sigma}{2}\right) \geq \frac{D}{2} \sin 1 > \frac{\sqrt{D}}{2}.$$

Hence $|PH| > 1$. This implies that the range circle of agent B must be at distance larger than 1 from the meeting area of w_A, which proves the lemma. ◀

New Distributed Interactive Proofs for Planarity: A Matter of Left and Right

Yuval Gil[1]
Reykjavík University, Iceland

Merav Parter
Weizmann Institute of Science, Rehovot, Israel

Abstract

We provide new distributed interactive proofs (DIP) for planarity and related graph families. The notion of a *distributed interactive proof* (DIP) was introduced by Kol, Oshman, and Saxena (PODC 2018). In this setting, the verifier consists of n nodes connected by a communication graph G. The prover is a single entity that communicates with all nodes by short messages. The goal is to verify that the graph G satisfies a certain property (e.g., planarity) in a small number of rounds, and with a small communication bound, denoted as the *proof size*.

Prior work by Naor, Parter and Yogev (SODA 2020) presented a DIP for planarity that uses three interaction rounds and a proof size of $O(\log n)$. Feuilloley et al. (PODC 2020) showed that the same can be achieved with a single interaction round and without randomization, by providing a proof labeling scheme with a proof size of $O(\log n)$. In a subsequent work, Bousquet, Feuilloley, and Pierron (OPODIS 2021) achieved the same bound for related graph families such as outerplanarity, series-parallel graphs, and graphs of treewidth at most 2. In this work, we design new DIPs that use exponentially shorter proofs compared to the state-of-the-art bounds. Our main results are:

- There is a 5-round protocol with $O(\log \log n)$ proof size for outerplanarity.
- There is a 5-round protocol with $O(\log \log n)$ proof size for verifying embedded planarity and $O(\log \log n + \log \Delta)$ proof size for general planar graphs, where Δ is the maximum degree in the graph. In the former setting, it is assumed that an embedding of the graph is given (e.g., each node holds a clockwise orientation of its neighbors) and the goal is to verify that it is a valid planar embedding. The latter result should be compared with the non-interactive setting for which there is lower bound of $\Omega(\log n)$ bits for graphs with $\Delta = O(1)$ by Feuilloley et al. (PODC 2020).
- The non-interactive deterministic lower bound of $\Omega(\log n)$ bits by Feuilloley et al. (PODC 2020) can be extended to hold even if the verifier is randomized. Moreover, the lower bound holds even with the assumption that the verifier's randomness comes in the form of an unbounded random string *shared* among the nodes.

We also show that our DIPs can be extended to protocols with similar bounds for verifying series-parallel graphs and graphs with tree-width at most 2. Perhaps surprisingly, our results demonstrate that the key technical barrier for obtaining $o(\log \log n)$ labels for all our problems is a basic sorting verification task in which all nodes are embedded on an oriented path $P \subseteq G$ and it is desired for each node to distinguish between its left and right G-neighbors.

2012 ACM Subject Classification Theory of computation → Interactive proof systems; Theory of computation → Distributed algorithms

Keywords and phrases Distributed interactive proofs, Planar graphs

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.34

Related Version *Full Version*: `https://arxiv.org/pdf/2505.00338` [14]

Funding This project is partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 949083.

[1] This work was done while the author was a postdoc at Weizmann.


© Yuval Gil and Merav Parter;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Planarity is a fundamental graph property that has been widely studied due to its rich combinatorial structure and numerous algorithmic applications. While in the centralized setting, the task of verifying if a given graph is planar can be done in linear time [18], in the distributed setting the running time depends linearly on the diameter of the graph [13]. The non-local nature of planarity motivates the use of a powerful, but potentially untrusted, *prover* that can aid the distributed verification by providing each node with a short auxiliary string, a.k.a. a *proof label*. The nodes then engage in brief communication to collectively determine whether to accept or reject the provided proof. This framework has been formalized into *proof labeling schemes* by Korman, Kutten and Peleg [20]. In this work we focus on the interactive extension of this model to *distributed interactive proofs* (DIP) as proposed by Kol, Oshman, and Saxena [19]. In this setting, the nodes are allowed to interact with the prover through multiple rounds of communication. The key complexity measures are the number of interaction rounds and the proof size.

The first evidence of the power of such proof systems for certifying planarity was provided by Naor, Parter, and Yogev [21]. Their result for planarity was in fact implied by a more general machinery that translates any (centralized) computation in $O(n)$ time – such as, the centralized planarity verification of [18] – into a three-round distributed interactive protocol with $O(\log n)$ proof size. Subsequent work by Feuilloley et al. [6, 7] demonstrated that the same proof size could be achieved with just a single interaction round, effectively reducing to the classical proof labeling scheme setting. Their work is accompanied by a matching lower bound of $\Omega(\log n)$ bits, that holds already for graphs with maximum degree of $O(1)$. These developments bring us back to the fundamental question of whether interaction truly provides an advantage in certifying planarity.

▶ **Question 1.1.** *What is the power of distributed interactive proofs for certifying planarity?*

We address this question by providing new DIPs for planarity and related graph families. Namely, we obtain constant-round protocols with a proof size of $O(\log \log n)$ for outerplanarity, embedded planarity, series-parallel graphs, and graphs of treewidth at most 2; and a proof size of $O(\log \log n + \log \Delta)$ for planarity in graphs of maximum degree Δ (we distinguish between embedded planarity in which we assume that a graph embedding is given in a distributed manner, and planarity in which no embedding is given; see full version [14] for formal definition). We also show that the $\Omega(\log n)$ lower bound of [6] can be extended to one-round DIPs. Therefore, our results give the first evidence to the advantage provided from interaction in planarity certification.

Model. In this paper, we consider *distributed interactive proofs (DIPs)* based on the model of [19].[2] In the DIP setting, instances are graphs $G = (V, E)$ taken from some universe $\mathcal{U}$ and the goal is to distinguish between *yes-instances* that come from a yes-family $\mathcal{F}_Y \subset \mathcal{U}$ and *no-instances* that come from a no-family $\mathcal{F}_N = \mathcal{U} - \mathcal{F}_Y$.[3] A DIP is an interactive protocol between a distributed *verifier* operating concurrently at all nodes of the graph and a centralized *prover* that can see the entire instance. The prover and verifier interact back and forth in *rounds*. Let $\mathcal{I}_{\texttt{vrf}}$ denote the rounds in which the verifier interacts with the prover and let $\mathcal{I}_{\texttt{prv}}$ denote the rounds in which the prover interacts with the verifier.

[2] We note that [19] uses the terms dAM and dMAM to denote the special cases of a DIP protocol with 2 and 3 rounds, respectively.

[3] One can easily adapt the setting so that instances also include some local information to the nodes (e.g., identifiers, weights, etc.). We chose to avoid this additional notation as the results of this paper apply to graphs without local node information.

Our protocols are *public-coin* which means that in each round $i \in \mathcal{I}_{\texttt{vrf}}$, the verifier at each node $v \in V$ interacts by drawing a random bitstring $\rho_v^i \in \{0,1\}^*$ and sending it to the prover (in particular, the verifier cannot hide any random bits from the prover). The prover interacts with the verifier in rounds $i \in \mathcal{I}_{\texttt{prv}}$ by sending a message $\mu_v^i \in \{0,1\}^*$ to each node $v \in V$. Keeping up with the terminology of [20], we sometimes refer to the messages sent by the prover as *labels*. The interaction ends with a round in which the prover interacts with the verifier, after which the verifier at each node $v \in V$ computes a local yes/no output based on: (1) the random bitstrings ρ_v^i drawn by v throughout the protocol; (2) the labels μ_v^i assigned to v by the prover throughout the protocol; and (3) the labels μ_u^i assigned to v's neighbors $u \in N(v)$ by the prover throughout the protocol. We say that the verifier *accepts* the instance if all nodes output "yes", and that the verifier *rejects* the instance if at least one node outputs "no".

As standard, the correctness of a proof system is defined by *completeness* and *soundness* requirements. The completeness requirements asks that if $G \in \mathcal{F}_Y$, then there exists an *honest* prover causing the verifier to accept the instance; whereas the soundness requirement asks that if $G \in \mathcal{F}_N$, then for any prover, the verifier rejects the instance. In the DIP setting, the correctness requirements are relaxed so that the completeness and soundness hold with probabilities $1-\epsilon_c$ and $1-\epsilon_s$, respectively, for some parameters $0 \leq \epsilon_c, \epsilon_s < 1/2$. In this case, we refer to ϵ_c as the *completeness error* and to ϵ_s as the *soundness error*. A protocol is said to have *perfect completeness* if $\epsilon_c = 0$. The performance of a DIP protocol is measured by the amount of prover-verifier communication it requires. Namely, the objective is to design protocols with a small number of interaction rounds and a small *proof size* which is defined as the size of the longest label assigned by the honest prover during the protocol.

The Challenge of Going Below the $\log n$ Barrier. As observed in [21], achieving sub-logarithmic proof lengths presents a significant challenge in the DIP setting and also serves as a lower bound for numerous problems in the non-interactive setting. This difficulty arises because many fundamental operations – such as identifying neighboring nodes, counting, or specifying node IDs – intrinsically require $\log n$ bits. While [21] made important initial progress in this area, their results apply to a more permissive variant of the DIP model, where nodes are allowed to send different messages to each of their neighbors. Indeed, their key technique is based on a rooted spanning tree provided by the prover such that every node identifies its tree-parent based on its internal port-numbering. Therefore, for each node to be able to learn its children in the tree (which is crucial to their protocols), every node has to send a distinct message to its parent.

In contrast, our work operates within the more restrictive DIP framework defined by Kol et al. [19], where nodes may only forward the proofs they receive to their neighbors. This constraint aligns with the non-interactive proof labeling model introduced in [20], where a node's decision is based solely on its own proof and those of its neighbors. This key difference in model assumptions becomes especially important in the sub-logarithmic setting, effectively preventing us from directly applying the techniques developed in [21].

Our Results. We present new distributed interactive proofs for various well-studied graph families. The first graph family considered is that of path-outerplanar graphs (see Section 2 for definition). Previously, [6] showed that path-outerplanarity admits a proof labeling scheme with a proof size of $O(\log n)$. We improve upon the communication complexity of that result by designing a protocol with exponentially shorter proof labels as specified in the following theorem.

▶ **Theorem 1.2.** *There exists a distributed interactive proof for path-outerplanarity running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

Building upon the path-outerplanarity protocol, we provide a protocol for (general) outerplanarity with the same asymptotic communication guarantees.

▶ **Theorem 1.3.** *There exists a distributed interactive proof for outerplanarity running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

We then move on to consider the case of planar graphs. In this context, we consider two verification tasks referred to as *planar embedding* and *planarity*. In the planar embedding task, an embedding of the graph is given in a distributed manner and the goal is to decide if it is a valid planar embedding (i.e., if no edges cross). In the planarity task, the goal is simply to decide if the given graph is planar. The details of our protocols for these tasks are given in the following two theorems.

▶ **Theorem 1.4.** *There exists a distributed interactive proof for planar embedding running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

▶ **Theorem 1.5.** *There exists a distributed interactive proof for planarity running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n + \log\Delta)$*.*

We also consider the two closely related graph families of series-parallel graphs and graphs of treewidth at most 2. We obtain the following two results.

▶ **Theorem 1.6.** *There exists a distributed interactive proof for series-parallel graphs running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

▶ **Theorem 1.7.** *There exists a distributed interactive proof for graphs of treewidth at most* 2 *running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

Finally, we provide the following lower bound.

▶ **Theorem 1.8.** *For each of the following graph families, any one-round distributed interactive proof with completeness and soundness errors smaller than* $1/10$ *requires a proof size of* $\Omega(\log n)$*: (1) path-outerplanar graphs; (2) outerplanar graphs; (3) embedded planar graphs; (4) planar graphs; (5) series-parallel graphs; and (6) graphs of treewidth at most* 2*;*

We note that Theorem 1.8 strengthens the lower bound presented in [6] in the following ways. First, the lower bound of [6] only applies to one-round proofs with *deterministic* verifier. Theorem 1.8 states that the same bound holds even if the verifier is *randomized*. Combined with the upper bounds stated above, our results present a strong evidence of the power added from *interaction* in the context of distributed proofs for planarity and related tasks. We remark that our lower bound holds even if the nodes have access to (unbounded) *shared* randomness. We also note that the lower bound of [6] does not explicitly apply to some of the graph families that appear in Theorem 1.8 (namely, path-outerplanar graphs, embedded planar graphs, and series-parallel graphs).

Open problems. Our results leave some intriguing unresolved questions that can be explored in follow-up works. Here, we highlight three of them.

As the main open problem, we ask whether the additive $O(\log \Delta)$ term is necessary in the proof size for planarity. That is, we pose the following question.

▶ **Open Question 1.** *Is it possible to obtain a constant round protocol for planarity with a proof size of $O(\log \log n)$ even on graphs with maximum degree $\Delta = \omega(\text{poly} \log n)$?*

One may also ask whether 5 interaction rounds are necessary in order to obtain a proof size of $O(\log \log n)$ for the tasks discussed in this paper. Of course, we know by Theorem 1.8 that 1 round is insufficient. However, for any $1 < r < 5$, whether an r-round protocol exists remains open even if we simply look for a proof size of $o(\log n)$. This leads to the following open problem.

▶ **Open Question 2.** *Is it possible to obtain an r-round protocol for e.g., outerplanarity, with a proof size of $o(\log n)$ for some $1 < r < 5$?*

Finally, we ask whether it is possible to improve our protocol's communication bound.

▶ **Open Question 3.** *Is it possible to obtain a protocol for e.g., outerplanarity, where the prover communicates $o(\log \log n)$ bits with each node?*

2 Preliminaries and Definitions

Conventions. Throughout, if not specified otherwise, a graph $G = (V, E)$ is assumed to be undirected and connected. For each node $v \in V$, we stick to the convention that $N_G(v)$ denotes the set of v's *neighbors* in the graph, $E(v)$ denotes the set of edges incident on v, and $\deg_G(v) = |N_G(v)| = |E(v)|$ denotes v's degree in G. Whenever G is clear from the context, we may omit it from the notation and write $N(v)$ and $\deg(v)$ instead of $N_G(v)$ and $\deg_G(v)$. For a node-subset $V' \subseteq V$, we denote by $G(V')$ the subgraph induced on G by V'.

As part of our technical tools, we also consider *directed graphs*. If G is directed, we assume that the edge orientation is given to the nodes such that each node $v \in V$ can distinguish between its incoming and outgoing incident edges. For a directed edge e with endpoints u and v, we write $e = (u, v)$ to reflect that e is directed from u to v, and $e = (v, u)$ otherwise. In the context of a distributed interactive proof, we assume that the label assigned by the prover to node $v \in V$ can be viewed by both its incoming and outgoing neighbors.

Hamiltonian paths. Consider a graph $G = (V, E)$ with a Hamiltonian path P. For a pair $u, v \in V$ of nodes, define the relation $\prec_P$ so that $u \prec_P v$ if u precedes v in P. Naturally, this extends to $u \preceq_P v$ if $u \prec_P v$ or $u = v$. Going forward, when P is clear from context, we may omit it from our notation and write $u \prec v$ and $u \preceq v$ instead of $u \prec_P v$ and $u \preceq_P v$, respectively. Whenever we encounter a Hamiltonian path, it will be convenient to think of it drawn as a straight line from left to right. Keeping up with this convention, for each node $v \in V$, we can partition its non-path edges in G into v-*left* edges which are incident on neighbors $u \prec v$, and v-*right* edges which are incident on neighbors $v \prec u$. We say that a non-path edge (u, v) is the *longest* v-left (resp., v-right) edge if $u \prec v$ (resp., $v \prec u$) and $u \prec u'$ (resp., $u' \prec u$) for every neighbor $u' \in N(v)$.

Left-right sorting. We define a verification task called *left-right sorting (LR-sorting)* which is used as a sub-task in our protocols. In LR-sorting, a directed graph $G = (V, E)$ is given. The graph G admits a directed Hamiltonian path P which is given such that each node

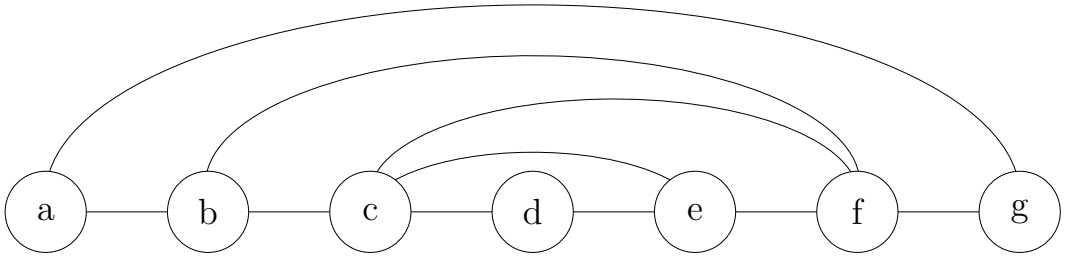


Figure 1 A path-outerplanar graph. The longest c-right edge is (c, f); the longest f-left edge is (b, f); the successor of (c, e) is (c, f).

$v \in V$ knows its incident edges in P. The path P is assumed to be directed from left to right. The goal of the task is to decide if $u \prec v$ for every directed edge $(u, v) \in E - P$. That is, a yes-instance is defined so that $u \prec v$ for every edge $(u, v) \in E$; whereas a no-instance admits at least one edge $(u, v) \in E$ such that $v \prec u$. Observe that equivalently, yes-instances are ones in which G is a DAG (in which case the P-ordering is the unique topological sort of G); and no-instances are ones in which G admits some cycle.

Path-outerplanar graphs. A graph $G = (V, E)$ is said to be *path-outerplanar* if it admits a Hamiltonian path P such that all non-path edges can be drawn above P without crossings. If the edges can be drawn in such a manner, we say that they are *properly nested* within P (or simply properly nested when P is clear from the context). Equivalently, a graph is path-outerplanar if no two edges $(u, v), (u', v') \in E$ satisfy $u \prec_P u' \prec_P v \prec_P v'$ with respect to some Hamiltonian path P (cf. [6]). Refer to Figure 1 for a pictorial example of a path-outerplanar graph and some of the related definitions.

The following simple observation will be useful in our protocol for path-outerplanar graphs in Section B.

▶ **Observation 2.1.** *Suppose that G is a path-outerplanar graph and let $(u, v) \in E$ be a non-path edge such that $u \prec v$. The edge (u, v) is either the longest u-right edge or the longest v-left edge.*

Proof. Assume that (u, v) is neither the longest u-right edge nor the longest v-left edge. Let (u, v') and (u', v) be the longest u-right and v-left edges, respectively. These edges satisfy $u' \prec u \prec v \prec v'$ which contradicts path-outerplanarity. ◀

Given a path-outerplanar graph G, we make the following definitions. For a non-path edge (u, v), $u \prec v$, define its *successor* as the edge (u', v') that satisfies: (1) $u' \preceq u \prec v \preceq v'$; and (2) $u'' \preceq u' \prec v' \preceq v''$ for every edge (u'', v'') that satisfies $u'' \preceq u \prec v \preceq v''$. Intuitively, the successor of an edge is the edge drawn directly above it. For cohesiveness, for any edge e that does not have a successor in the graph, define the successor to be a virtual edge $e^* = (u^*, v^*)$, $u^*, v^* \notin V$, defined so that $u^* \prec v \prec v^*$ for any $v \in V$. Notice that each edge has a unique successor. Naturally, we say that e is a *predecessor* of e' if e' is the successor of e. We say that two edges e and e' are *siblings* if they have a common successor.

The following observation is now straightforward from the definitions.

▶ **Observation 2.2.** *Suppose that G is a path-outerplanar graph and let $e = (u, v)$ be a (possibly virtual) non-path edge such that $u \prec v$. There exists an ordering $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ of e's predecessors such that $u \preceq u_1 \prec v_1 \preceq u_2 \prec v_2 \cdots \preceq u_k \prec v_k \preceq v$.*

Encoding a spanning forest in a planar graph. As a building block in our protocol, we would like for the prover to be able to communicate a spanning forest F of the graph G to the verifier. While it is trivial to achieve in general using $O(\log n)$-bit labels, in our case we would like much smaller labels. It turns out that this task can be achieved in planar graphs deterministically and with constant-sized labels. This is done by slightly extending a construction of [3] which is designed for the task of deciding whether a planar graph admits a perfect matching.[4] We state the construction's properties in the following lemma.

▶ **Lemma 2.3.** *Let G be a planar graph and let F be a rooted spanning forest of G (i.e., F is a collection of rooted trees). For some constant $c > 0$, there exists a label assignment $L : V \to \{0,1\}^c$ such that each node $v \in V$ can learn its parent and children in F only as a function of $L(v)$, and the labels $L(u)$ assigned to v's neighbors $u \in N(v)$.*

For completeness of presentation, we provide a proof for the lemma. We emphasize that this construction only allows the prover to communicate the forest F to the verifier and does not provide proof that F is indeed a spanning forest.[5]

Proof. For ease of presentation, let us assume that the prover tries to communicate a spanning tree T (i.e., a connected forest). The case of an unconnected forest admits a similar construction. Suppose that $G = (V, E)$ is a planar graph and T is a spanning tree rooted at some node $r \in V$. For each node $v \in V - \{r\}$, let $\mathsf{parent}(v)$ denote its parent and let $\mathsf{depth}(v)$ denote its depth. Define the graph G_{odd} (resp., G_{even}) to be the graph obtained by starting from G and contracting all edges $(v, \mathsf{parent}(v))$ that go from an odd (resp., even) depth node v to its parent in T. Observe that G_{odd} and G_{even} are both planar and thus, 4-colorable. Towards providing the label assignment, the prover computes 4-colorings of G_{odd} and G_{even}, respectively. For each node $v \in V$, let $c_1(v)$ the color of the node into which v contracted in G_{odd} and let $c_2(v)$ be the color of the node into which v contracted in G_{even}. The prover assigns each node $v \in V$ with the label $L(v) = (c_1(v), c_2(v), \mathtt{parity}(v))$ where $\mathtt{parity}(v) = \mathsf{depth}(v) \bmod 2$.

We argue that the label assignment allows each node $v \in V$ to deduce which of its neighbors are its parent and children in T. The idea is as follows. If a node $v \in V$ of odd depth receives a color $c_1(v)$, then due to the validity of the coloring on G_{odd}, it holds that $\mathsf{parent}(v)$ is the only neighbor of v with even depth for which $c_1(\mathsf{parent}(v)) = c_1(v)$. The case of even depth nodes is similar.

To make things more concrete, consider some node $v \in V$ with $\mathtt{parity}(v) = 1$ (resp., $\mathtt{parity}(v) = 0$). Node v identifies its parent as its only neighbor $u \in N(v)$ with $\mathtt{parity}(u) = 0$ and $c_1(v) = c_1(u)$ (resp., $\mathtt{parity}(v) = 1$ and $c_2(v) = c_2(u)$). Additionally, v identifies its children as the neighbors $u \in N(v)$ that satisfy $\mathtt{parity}(u) = 0$ and $c_2(v) = c_2(u)$ (resp., $\mathtt{parity}(v) = 1$ and $c_1(v) = c_1(u)$). ◀

Enabling edge-labels in planar graphs. In the technical sections, it will be convenient to describe protocols assuming that the prover can also assign edge-labels (such that both of the edge endpoints can see the label) rather than only node-labels. This assumption is facilitated by the following lemma.[6]

[4] The scheme extends to some classes of non-planar graphs; see [3] for full details.

[5] Another way to formulate this construction is in terms of *advice*, based on the model of [9, 10]. Specifically, using the terminology of [10], the statement means that computing any spanning forest of a planar graph admits an $(O(1), 0)$-advising scheme.

[6] Transformations that enable edge-labels in planar graphs have been presented in previous papers (see, e.g., [6]). However, these constructions require the prover to assign an ordering to the nodes which

▶ **Lemma 2.4.** *Let Π be a class of planar graphs. Suppose that there exists a distributed interactive proof deciding whether $G \in \Pi$ in which the prover assigns labels of size ℓ to the nodes and edges. Then, there exists a distributed interactive proof in which the prover assigns labels of size $O(\ell)$ only to the nodes. Furthermore, the two proofs admit the same number of interaction rounds.*

Proof. It is well-known that planar graphs have arboricity at most 3. This means that the edge-set of any planar graph $G = (V, E)$ can be partitioned into three edge-disjoint forests F_1, F_2, F_3. By Lemma 2.3, the prover can inform each node $v \in V$ of its parent and children in each F_i using only constant-sized labels. Then, instead of assigning a label $L(u_i, v)$ to the edge (u_i, v) between v and its parent u_i in F_i, the prover simply writes $L(u_i, v)$ to a field in v's label which is designated for its parent in F_i. This allows both endpoints to learn the label $L(u_i, v)$, thus enabling the simulation of edge-labels. ◀

Spanning tree verification. Consider a graph $G = (V, E)$ and let T be a subgraph of G such that each node $v \in V$ knows its incident edges in T. We define *spanning tree verification* as the task of deciding whether T is a spanning tree of G. The following lemma is established in [21].

▶ **Lemma 2.5** ([21, Section 7.1])**.** *There is a distributed interactive proof for spanning tree verification with 3 interaction rounds and constant proof size. The proof admits perfect completeness and a constant soundness error.*

Observe that by standard parallel repetition, one can reduce the soundness error to $1/2^\ell$ at the expense of a $\Theta(\ell)$ proof size for any parameter $\ell > 0$. Throughout the paper, this fact will be used in a black-box manner.

Multiset equality. In the *multiset equality* problem, each node $v \in V$ receives as input two multisets $S_1(v), S_2(v)$ and the goal is to decide whether $S_1 = S_2$, where S_1 and S_2 are the multisets $S_1 = \bigcup_{v \in V} S_1(v)$ and $S_2 = \bigcup_{v \in V} S_2(v)$. Notice that in the definition of S_1 and S_2, the union is taken with respect to multisets, i.e., the multiplicity of element s in S_1 (resp., S_2) is the sum of its multiplicities over all multisets $S_1(v)$ (resp., $S_2(v)$). The multisets S_1 and S_2 are assumed to be of size at most k for some integer $k > 0$, and the elements are taken from a universe of size k^c for some constant $c \geq 1$. For our purposes, it would also be convenient to assume that the nodes are given a distributed encoding of a rooted spanning tree of the graph. The following lemma can be derived from the multiset equality protocol of [21].

▶ **Lemma 2.6** ([21])**.** *Given a multiset equality instance (G, S_1, S_2, k) such that $|S_1|, |S_2| \leq k$ and a rooted spanning tree T of G, there exists a 2-round distributed interactive proof for multiset equality. The proof admits perfect completeness, a soundness error of $1/k^c$, and a proof size of $O(\log k)$.*

Since the lemma above is not explicit in [21] and since we use the details of the multiset equality protocol in a white-box manner, we describe here the construction's details. The multiset equality protocol relies on the following idea. For a multiset S, define the polynomial $\varphi_S(x) = \prod_{s \in S}(s - x)$.[7] Now, observe that $S_1 = S_2$ if and only if $\varphi_{S_1} \equiv \varphi_{S_2}$. Moreover,

incurs an additive $\Theta(\log n)$ overhead to the label size. Thus, we cannot use these transformations for our purposes.

[7] Notice that we assume here w.l.o.g. that multiset elements are integers.

notice that the degree of $\varphi_{S_1}(x)$ and $\varphi_{S_2}(x)$ is at most k. Define p as the smallest prime number that satisfies $p > k^{c+1}$ and let z be a variable drawn uniformly at random from $\{0, \ldots, p-1\}$. By polynomial identity testing properties, if $\varphi_{S_1} \not\equiv \varphi_{S_2}$ and $\varphi_{S_1}(z), \varphi_{S_2}(z)$ are computed over the field $\mathbb{F}_p$, then $\Pr[\varphi_{S_1}(z) = \varphi_{S_2}(z)] \leq k/p \leq 1/k^c$.[8]

Following this idea, the multiset equality problem essentially reduces to evaluating a polynomial at a random point $z \in \{0, \ldots, p-1\}$. Recalling that we assume that a distributed encoding of a rooted spanning tree is provided to the nodes, the polynomial evaluation is implemented as follows. First, the point $z \in \{0, \ldots, p-1\}$ is sampled by the root and sent to the prover. Then, the prover assigns each node $v \in V$ with the value z and the values $\varphi_{S_1^v}(z), \varphi_{S_2^v}(z)$ (computed over $\mathbb{F}_p$) where S_1^v (resp., S_2^v) is the multiset of elements from S_1 (resp., S_2) in v's subtree. Given the assigned values, it is well-known that their validity can be checked at each node $v \in V$ based on its input, its label, and its children's labels. This is because polynomial evaluation is an aggregation task that can be verified "up the tree" (see, e.g., [20, Lemma 4.4] for details). Following these checks, the root r can check that $\varphi_{S_1^r}(z) = \varphi_{S_2^r}(z)$ (which implies that $\varphi_{S_1}(z) = \varphi_{S_2}(z)$).

To summarize, given a rooted spanning tree of the graph, the multiset equality protocol runs for 2 interaction rounds, admits perfect completeness, a soundness error of $1/k^c$, and a proof size of $O(\log p) = O(\log k)$.[9]

3 Technical Overview

In this section, we provide an overview of the techniques used to obtain the protocols presented in the paper. To exemplify the challenge of planarity certification with labels of size $O(\log \log n)$, let us first sketch a seemingly natural (yet unsuccessful) approach to which we refer as the *clustering* approach. Suppose that the prover computes a partition of the graph into node-disjoint connected clusters of size $\operatorname{poly} \log n$. Then, the prover provides a proof that: (1) the subgraph induced by each cluster is planar; and (2) the graph obtained by contracting all clusters is planar. Notice that in terms of *proof size* this approach is promising (and indeed, a similar approach was used in [21] to achieve sub-logarithmic proofs for various other problems). This is because one can use , e.g., the logarithmic proof for planarity of [6] on each cluster to obtain a proof of size $O(\log \log n)$ for (1). As for (2), since each cluster has size $\operatorname{poly} \log n$ and acts as a single node in the contracted graph, one can hope that it is possible to distribute the proof of a single cluster among the nodes within that cluster using only $O(\log \log n)$-sized labels per node. For the sake of this example, let us assume that it is indeed possible to obtain a proof for (2) with a proof size of $O(\log \log n)$.

It is not hard to see that the proof provided from the clustering approach is complete – if G is planar, then so are the subgraphs induced by the clusters as well as the graph obtained from contracting every cluster. The fundamental problem however, is that a proof for planarity obtained from this approach *cannot* be sound. To see why this is the case, consider a non-planar graph that contains a single 5-clique H.[10] A cheating prover can then define the partition such that, e.g., two nodes of H are assigned to one cluster and the other three are assigned to a different cluster. In this case, H does not violate planarity within any cluster and translates to a single edge in the contracted graph. Therefore, in this instance the verifier is likely to accept which violates the soundness requirement.

[8] Recall that $\mathbb{F}_p$ is the field whose elements are $\{0, \ldots, p-1\}$ and operations are done modulo p.
[9] Recall that standard density of primes properties assure that p cannot be too large. In particular, $p < k^{c+2}$, and thus $\log p = O(\log k)$.
[10] Recall that a graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

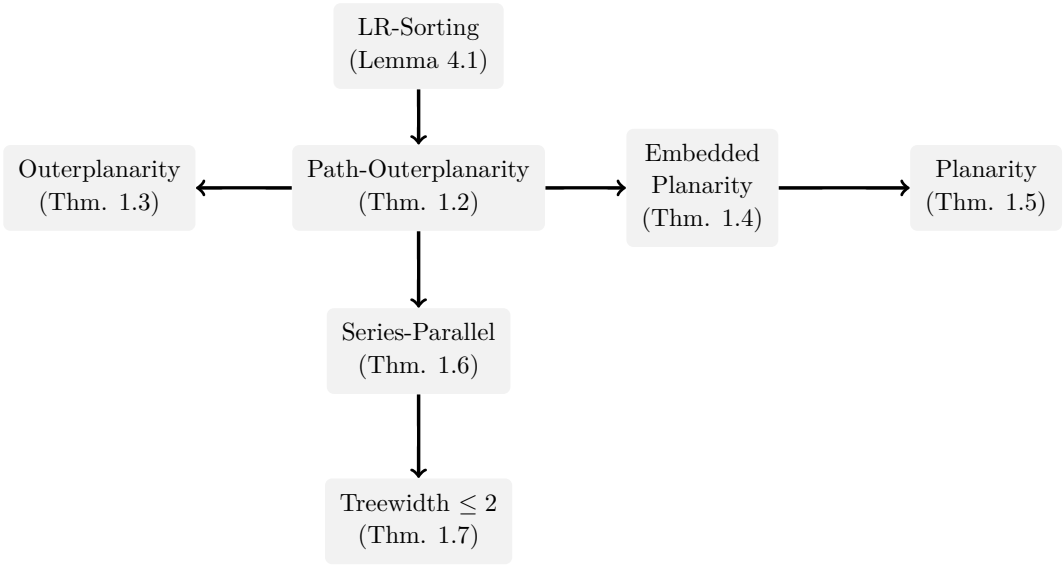


Figure 2 High-level description of the main results and their connections.

Notice that even if somehow we were able to prevent the prover from constructing an adversarial partition, it is possible to construct a no-instance in which the verifier is likely to accept for *any* partition. For example, it is possible to create an instance where each edge of H is subdivided such that its endpoints are at distance $\Omega(n)$ in G (and thus, are separated in any partition). Hence, we conclude that the clustering approach is doomed to fail for the planarity task.

The inherent failure of the clustering approach compels us to come up with a different approach. Similarly to the approach of [6], we seek to reduce planarity tasks to some other well-structured task for which we are able to design an efficient protocol. Perhaps surprisingly, we show that efficient protocols for all tasks considered in the current paper can be obtained based on a protocol for the seemingly unrelated task of LR-sorting. Indeed, starting from LR-sorting we present a sequence of reductions leading to new protocols for outerplanarity, planar embedding, planarity, series-parallel, and graphs of treewidth ≤ 2. Refer to Figure 2 for a chart depicting the dependencies between our different constructions. Notice that an advantage of LR-sorting is that unlike the planarity task, it can benefit from the clustering approach. As we explain below in more detail, this becomes useful in our protocol.

LR-sorting. To give an intuition for the LR-sorting protocol, let us first sketch a simple one-round protocol for LR-sorting with a proof size of $O(\log n)$. The prover assigns each node with its position on the path. Then, each node $v \in V$ that is located at the i-th position can verify that: (1) its path-neighbors are located at positions $i \pm 1$; and (2) all of its outgoing edges go towards nodes of larger positions.

To obtain a distributed interactive proof with a proof size of $O(\log \log n)$, the idea is to divide the nodes into node-disjoint *blocks* where each block is made up of $\lceil \log n \rceil$ consecutive path-nodes. This allows the prover to distribute the position of block b, denoted by $\texttt{pos}(b)$, such that each node receives: (1) its *index* $i \in [\lceil \log n \rceil]$ within b; and (2) $\texttt{pos}(b)[i]$, i.e., the i-th most significant bit of b's position. Ideally, as in the clustering approach, we would like for each block to act as a single node in the trivial protocol. In this short description, let us assume for simplicity that the prover encodes the position of the blocks correctly according

to the P-ordering and let $\texttt{pos}(b)$ denote the position of block b. The main challenge of the protocol is then captured by the following question: suppose that there is an edge (u, v) where u and v belong to different blocks b_u, b_v, how can the prover prove to u and v that $\texttt{pos}(b_u) < \texttt{pos}(b_v)$?

Towards answering this question, let us first consider the simpler case where (u, v) is the *only* edge leaving the block for both b_u and b_v. Furthermore, we will describe the protocol under the assumption that the prover can assign edge-labels. Recall that Lemma 2.4 implies that the edge-labels assumption can be simulated in planar graphs while incurring only a constant overhead to the proof size. Let i be the index of the most significant bit in which $\texttt{pos}(b_u)$ and $\texttt{pos}(b_v)$ differ. Notice that by definition, if $\texttt{pos}(b_u) < \texttt{pos}(b_v)$, then $\texttt{pos}(b_u)[i] = 0$ and $\texttt{pos}(b_v)[i] = 1$. In addition, i is in the range $[\lceil \log n \rceil]$ and thus, can be encoded using $O(\log \log n)$ bits. In the first interaction, the prover encodes i to the label of (u, v). The prover then needs to prove that (1) $\texttt{pos}(b_u)[i] = 0$; (2) $\texttt{pos}(b_v)[i] = 1$; and (3) $\texttt{pos}(b_u)$ and $\texttt{pos}(b_v)$ agree on their $i - 1$ most significant bits.

Let us focus on how (3) is proved. A straightforward way to obtain a proof for (3) is to assign the edge (u, v) with the substring containing the $i - 1$ most significant bits in their blocks. The main problem with this approach is that the proof size could be as large as $\Theta(\log n)$. To avoid this large label size, we can use polynomial identity testing. That is, we interpret the substrings containing the $i - 1$ most significant bits in $\texttt{pos}(b_u)$ and $\texttt{pos}(b_v)$ as polynomials of degree $O(\log n)$ and seek to have the prover and verifier evaluate them at a random point over a field of size $\operatorname{poly} \log n$. The prover then assigns the outcome of the polynomial evaluation to the label of (u, v). This introduces the following *locality* problem: the verification that the polynomials were computed correctly is done at the $(i - 1)$-th leftmost node in each block (using the standard aggregation technique of [20]). Since u and v might be far from the $(i - 1)$-th leftmost node in their respective blocks, they cannot locally detect that the prover is not lying about the outcome. We note that in the simplified case where (u, v) is the only edge leaving b_u and b_v, the problem is easy to solve. Indeed, since (u, v) is the only edge considered by blocks b_u and b_v, the prover can assign the outcome of the polynomial evaluation at the $(i - 1)$-th node to *all* nodes of b_u and b_v. The nodes can then check the correctness of this assignment locally based on the assignment to their block-neighbors.

Moving on to the general case where each block may have *many* edges leaving it, it is not clear how to solve the locality problem described above. Of course, if the prover assigns each node of the block with the outcomes of polynomial evaluations for every relevant index, this could require assigning $\Theta(\log n)$ values to every node which completely defeats the purpose. The main observation that we make is that one can formulate the locality problem as an instance of *multiset equality* between two multisets that are carefully defined within each block. Then, to check whether the multisets are indeed equal, a multiset equality protocol is executed within the blocks. An important property of the constructed multisets is that they are of size $\operatorname{poly} \log n$ which means that this final step can be done while maintaining a proof size of $O(\log \log n)$.

For the correctness, it turns out that the protocol's completeness becomes straightforward from the multisets construction, whereas the soundness argument requires a bit more care. Essentially, we show that in a no-instance the prover is likely to *commit* to a pair of unequal multisets within some block. Then, conditioning on this event, it is likely that the verifier rejects the instance due to the soundness of the multiset equality protocol.

From LR-sorting to path-outerplanarity. In Section B, we devise a protocol for path-outerplanarity. To get intuition of how the protocol works, we first recall the labels assigned in the non-interactive proof of [6]. Essentially, the prover assigns each node $v \in V$ with its position in the path P along with the position of the nodes u and u' for which $(u, u') \in E$ is the first edge that is drawn above v. The authors of [6] show that this labeling allows the (deterministic) verifier to verify that indeed G is path-outerplanar.

Of course, assigning the positions of nodes in P is far too costly for our purposes, so we seek to avoid it by using interaction and randomization. To that end, we first note that the assignment of positions in [6] serves the following purposes: (1) each node learns its P-neighbors; (2) each edge can be identified based on the positions of its endpoints; and (3) each node learns a clockwise orientation of its incident edges. Note that since P is a spanning tree, (1) can be obtained using only constant-sized labels based on Lemmas 2.3 and 2.5. For (2), we show that the edge identifiers can be replaced by random bits. As for (3), we design a protocol in which it is sufficient for every node to only distinguish between its left and right edges with respect to the path P. In fact, we show that under the assumption that every node knows its left and right edges, path-outerplanarity can be solved in 3 rounds and with a *constant* proof size (refer to Lemma B.1 for the full details of the reduction). To lift this assumption, we can apply our LR-sorting protocol which leads to the stated complexity bound of Theorem 1.2.

From path-outerplanarity to outerplanarity and planarity. We handle outerplanarity and planarity through reductions to path-outerplanarity. We note that while such reductions are presented in previous works ([6] for planarity; [2] for outerplanarity), we cannot use them as-is in our setting. This is because these reductions incur an additive $\Theta(\log n)$ overhead to the proof size. Nevertheless, our results rely on some modifications of the existing reductions. For outerplanarity, our reduction is white-box and it avoids the $\Theta(\log n)$ overhead based on the tools of Lemma 2.3 and Lemma 2.5 as well as some observations regarding the path-outerplanarity protocol.

For planarity, we start from the planar embedding task as an intermediate point. To reduce planar embedding to path-outerplanarity, we revisit some of the constructive proofs presented in [6] and show that they can be used to obtain such a reduction. Once again, Lemma 2.3 and Lemma 2.5 are used for the sake of efficient implementation. Then, we show that planarity reduces to planar embedding while incurring only an additive $O(\log \Delta)$ overhead to the proof size, thus obtaining the stated result.

Lower bounds. The starting point for our lower bounds is the lower bound of [6] which applies to proof labeling schemes (in fact, it holds also for the more general locally checkable proofs [16]). The result of [6] shows that an $\Omega(\log n)$ proof size is required even for the task of deciding whether a graph is outerplanar or non-planar. We start by adjusting the lower bound's details so that it would apply for the task of deciding whether a graph is *biconnected* outerplanar or non-planar.[11] This adjustment leads to a bound for all the graph families considered in the current paper since they are planar and contain all biconnected outerplanar graphs. To extend the lower bound to one-round protocols in which the verifier is randomized, we use a framework presented in [11].

[11] Recall that a graph is biconnected if the removal of any node leaves the resulting graph connected. A biconnected component of a graph G is a maximal biconnected subgraph of G.

4 LR-Sorting Protocol

In this section, we present a protocol for the task of LR-sorting on a given directed graph $G = (V, E)$ with Hamiltonian path P. The protocol is implemented under the assumption that the prover is able to assign labels to the nodes and the edges of G. If a label $L(u, v)$ is assigned to the edge $(u, v) \in E$, then both endpoints u and v can view it. The main result of the current section is the following lemma.

▶ **Lemma 4.1.** *There exists a distributed interactive proof for LR-sorting running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$ *where labels are assigned to both nodes and edges.*

Recall that by Lemma 2.4, if the given graph is planar, we can lift the edge-labels assumption of Lemma 4.1 to get the following.

▶ **Lemma 4.2.** *There exists a distributed interactive proof for LR-sorting in planar graphs running in* 5 *interaction rounds. The proof admits perfect completeness, a soundness error of* $1/\text{poly}\log n$*, and a proof size of* $O(\log\log n)$*.*

The rest of the section is dedicated to the protocol's description. Recall that our goal is to show how the prover proves that $u \prec v$ for every non-path edge (u, v) directed from u to v. This is described in two stages. First, a division of the path into node-disjoint *blocks* is described. Then, we explain how the block construction allows the nodes to compare their relative position on the path. For clarity, the stages are described without regard for the number of interaction rounds. Then, in Appendix A, we explain how the protocol can be implemented in 5 interaction rounds. Throughout, $c > 0$ is defined as a positive constant that can be made large enough to support the protocol's soundness guarantee.

4.1 The Block Construction

The block construction is defined so that the first block consists of the $\lceil \log n \rceil$ leftmost nodes in the path, the second block consists of the next $\lceil \log n \rceil$ nodes and so on. For ease of presentation, we assume that all blocks are of size exactly $\lceil \log n \rceil$. One can easily adjust the protocol's details to handle the general case in which (only) the rightmost block may have more than $\lceil \log n \rceil$ (but less than $2\lceil \log n \rceil$) nodes.

The purpose of the block construction is to allow the nodes to receive information regarding their position on the path. The position of a block b, denoted by $\texttt{pos}(b)$, is defined to be $i - 1$ if b is the i-th leftmost block. Notice that due to the block size, it is possible to encode an integer $x \in \{0, \ldots, n-1\}$ through the nodes of a block using only $O(\log\log n)$ bits per node. To do so, assign the j-th leftmost node of the block with the number j as well as the j-th most significant bit of x (leading zeros are added if necessary). Using this mechanism, the prover assigns the values $\texttt{pos}(b)$ and $\texttt{pos}(b) + 1$ to each block b.

In addition, the prover provides a proof that the two numbers assigned to each block are consecutive. To explain how this is done, suppose that x is a nonnegative integer in binary representation and let j be its least significant bit valued 0. Notice that x and $x + 1$ differ (only) in their j least significant bits. For a block b, define j_b to be the least significant bit in $\texttt{pos}(b)$ whose value is 0 and let v_b be the node associated with the index j_b. To prove that the numbers assigned to b are consecutive, the prover marks v_b and informs every other node in the block whether it is to the right/left of v_b.

To present the verification process at block b, let us denote by $x_1(b)$ and $x_2(b)$ the bitstrings assigned to b under the claim $x_1(b) + 1 = x_2(b)$. If a node $v \in b$ was labeled to be to the right of v_b, then it checks that its bit in $x_1(b)$ is 1, its bit in $x_2(b)$ is 0, and its right

neighbor in the block (if such neighbor exists) is also labeled as right of v_b; if v was marked as v_b, then it checks that its bit in $x_1(b)$ is 1, its bit in $x_2(b)$ is 0, its right neighbor is labeled as right of v_b, and its left neighbor is labeled as left of v_b; and if v is labeled as left of v_b, then it checks that it received the same bit in both bitstrings and that its left neighbor is labeled as left of v_b.

To complete the block construction stage, the verifier checks that the position assignment is consistent between adjacent blocks. Let b and b' be two adjacent blocks where b' is to the right of b. The verifier seeks to check that $\texttt{pos}(b) + 1 = \texttt{pos}(b')$. To that end, the multiset equality protocol is used between $x_2(b)$ and $x_1(b')$, where a bitstring is interpreted as the subset of $[\lceil \log n \rceil]$ that contains the indices whose bit is 1. Notice that the sets (and so, the degree of the multiset equality polynomials) are of size at most $\lceil \log n \rceil$. The verifier and prover run the multiset equality protocol over the field $\mathbb{F}_p$ where p is the smallest prime satisfying $p > \log^c n$. The polynomials are computed at a random point $r \in \{0, \ldots, p-1\}$ which is the same for all blocks. To that end, the variable r is sampled by the leftmost node in the path and passed to all nodes in the graph by the prover. Each block b computes (with the prover's assistance) the values of the two polynomials associated with its encoded bitstrings $x_1(b), x_2(b)$. This allows every pair of adjacent blocks to check that the positions assigned to them are indeed consecutive.

Correctness. If the position assignment given by the prover is valid, then $x_2(b) = x_1(b')$ for every pair of adjacent blocks b, b' and by the completeness of the multiset equality protocol, the verifier does not reject in this case. On the other hand, if the position assignment is not valid, then at least one pair b, b' of adjacent blocks satisfies $x_2(b) \neq x_1(b')$ and by the soundness of the multiset equality protocol, the verifier rejects with probability $1 - \lceil \log n \rceil / p = 1 - 1/\text{poly} \log n$.

Remark. An alternative approach to verifying the validity of the block construction is to use the RAM compiler of [21] concurrently on pairs of consecutive blocks. Nevertheless, the approach and notations presented above will be useful in the presentation of the next stage. We also note that it might be plausible to implement the block construction stage with proof size of $o(\log \log n)$, we avoided these optimizations as the key "communication bottleneck" lies in the next stage.

4.2 Comparing Relative Positions

We now describe how the prover uses the block construction to prove claims of the form $u \prec v$ for all non-path edges (u, v). To that end, we divide the edges into two types as follows. The *inner-block* edges are defined as the edges (u, v) in which u and v belong to the same block, and the *outer-block* edges are defined as the edges (u, v) in which u and v belong to different blocks.

Inner-block edges. Suppose that (u, v) is an inner-block edge. To show that $u \prec v$, the prover first assigns a bit to the edge (u, v) indicating that it is an inner-block edge. Let us denote the indices of u and v within their block by i_u and i_v, respectively (recall that these indices were assigned to the nodes during the block construction stage). The nodes u and v check that $i_u < i_v$ and if not, reject immediately. If $i_u < i_v$, then it is left to check that u and v are indeed on the same block. To that end, the leftmost node of each block b (i.e., the node associated with the most significant bit of $\texttt{pos}(b)$) samples a number $r_b \in [\log^c n]$ and sends it to the prover which in response, sends the value r_b to all nodes of the block b.

Each node checks that the number it received is consistent with its block neighbors and the leftmost node in the block checks that it received the same number it sampled. Then, for every edge (u, v) that was labeled as inner-block, u and v check that they both received the same r_b value and reject otherwise.

Correctness for inner-block edges. For completeness, observe that if $u \prec v$, then all checks succeed and the verifier accepts. For soundness, if $v \prec u$ and u and v are on the same block, then it must hold that $i_v < i_u$ and the verifier rejects. So, suppose that v and u are on different blocks $b \neq b'$ but the prover labels (u, v) as an inner-block edge. Then, the verifier rejects unless $r_b = r_{b'}$ which happens with probability $1/\text{poly}\log n$.

Outer-block edges. We complete the protocol's description by addressing the case of outer-block edges. Consider an outer-block edge (u, v), i.e., u and v belong to different blocks b_u and b_v, respectively. The prover's goal is to show that $\texttt{pos}(b_u) < \texttt{pos}(b_v)$. We divide the proof into two parts referred to as the *commitment* scheme and the *verification* scheme.

The main idea behind the commitment scheme relies on the following simple fact. Suppose that x and y are two nonnegative integers represented by binary strings of same length (leading zeros are added if necessary). Then, $x < y$ if and only if there exists an index i such that the $i-1$ most significant bits of x and y are identical, the i-th most significant bit of x is 0, and the i-th most significant bit of y is 1. We shall refer to this index as the (x, y)-*distinguishing* index and denote it by $I(x, y)$.

Consider a non-path edge (u, v) whose endpoints belong to different blocks b_u and b_v, respectively. The commitment scheme starts by having the prover write the value $I_{u,v} = I(\texttt{pos}(b_u), \texttt{pos}(b_v))$ to the label of edge (u, v). Then, for each block b, the multiset equality polynomial that is associated with $\texttt{pos}(b)$ is computed at a random point $r' \in \{0, \ldots, p-1\}$, where p is the prime number defined above. Similarly to the block construction stage, the computation is done over the finite field $\mathbb{F}_p$ and the variable r' is the same for all blocks. For an index $i \in [\lceil \log n \rceil]$, let $\texttt{pos}(b)[1, \ldots, i]$ denote the substring of $\texttt{pos}(b)$ consisting of its i most significant bits and let us denote by φ_i^b the multiset equality polynomial that is identified with the substring $\texttt{pos}(b)[1, \ldots, i]$. We note that $\varphi_i^b(r')$ is exactly the value computed at the i-th leftmost bit of block b. In addition to computing the multiset equality polynomial values within the blocks, the prover writes the value $\varphi^{b_u}_{I_{u,v}-1}(r')$ (which is equal to $\varphi^{b_v}_{I_{u,v}-1}(r')$ by the definition of the distinguishing index) to the label of each non-path edge (u, v).

For an edge $e = (u, v)$ which is classified as an outer-block edge, let $\rho(e) = (i, j)$ be the pair of values assigned by the prover on the label of e during the commitment scheme. That is, here i is claimed by the prover to be the distinguishing index between u and v's block positions, and j is claimed to be the multiset equality polynomial value computed at the $(i-1)$-th index of both blocks. To complete the commitment scheme, the verifier at each node $v \in V$ makes some consistency checks. First, if the same index i appears as the first element in two pairs $\rho(u, v)$ and $\rho(v, u')$ associated with edges $(u, v), (v, u')$, then the verifier rejects. To see why this condition is imposed, notice that $u \prec v$ requires that the i-th bit of v's block is 1, whereas $v \prec u'$ requires that the i-th bit of v's block is 0. For the second consistency check, the verifier checks that if two of v's incident edges agree on the first element of $\rho(\cdot)$ (and did not fail the first check), then they agree on the second element of $\rho(\cdot)$. For each node $v \in V$, let us define $C_0(v)$ (resp., $C_1(v)$) as the set of pairs $\rho(u, v)$ (resp., $\rho(v, u)$) assigned by the prover to the edges (u, v) (resp., (v, u)) during the commitment scheme. Notice that we define $C_0(v), C_1(v)$ as sets and not multisets. In particular, this means that $|C_0(v)| + |C_1(v)| \leq \lceil \log n \rceil$.

The purpose of the verification scheme is to verify the validity of the values in $C_0(v)$ and $C_1(v)$ for each node $v \in V$. Notice that this cannot be achieved locally in a trivial manner since the indices that appear in $C_0(v)$ and $C_1(v)$ may be associated with nodes on v's block that are not adjacent to v. We describe the verification of $C_1(v)$ values and then explain the small change required for the $C_0(v)$ verification. For a block b, define $C_1(b)$ as the *multiset* $C_1(b) = \bigcup_{v \in b} C_1(v)$. Define $F(b)$ as the set of indices whose bit in $\texttt{pos}(b)$ is equal to 1 and let $D_1(b) = \bigcup_{i \in F(b)} \{(i, \varphi^b_{i-1}(r'))\}$. The main idea behind the verification scheme is that in yes-instances, for each node $v \in b$ and pair $(i,j) \in C_1(v)$, it holds by construction that $(i,j) \in D_1(b)$. Thus, one can construct a multiset which is equal to $C_1(b)$ by taking every element of $D_1(b)$ with some multiplicity between 0 and $\lceil \log n \rceil$ (notice that it is not guaranteed that every element of $D_1(b)$ is in $C_1(b)$, so we allow a "multiplicity" of 0). Following this idea, the validity of $C_1(b)$ can be verified by means of another multiset equality protocol.

To make things more concrete, consider a node $v \in b$ which is associated with an index $i_v \in F(b)$. The prover provides v with a value $M_v \in \{0, \ldots, \lceil \log n \rceil\}$ that counts the number of times the pair $(i_v, \varphi^b_{i_v - 1}(r'))$ appears in $C_1(b)$. Then, the prover and verifier execute a multiset equality protocol to compare between $C_1(b)$ and the multiset obtained by taking M_v copies of the pair $(i_v, \varphi^b_{i_v-1}(r'))$ for each $i_v \in F(b)$. Here, notice that node v gets the value i_v from its own label and the value $\varphi^b_{i_v-1}(r')$ from the label of its left neighbor on the path. When computing the multiset equality polynomials, each pair $(i,j) \in [\lceil \log n \rceil] \times \{0, \ldots, p-1\}$ is mapped to an element from the set $[p \cdot \lceil \log n \rceil]$ by means of a fixed bijection known in advance to all nodes. To accommodate this range of field elements, it suffices to execute the multiset equality protocol over the field $\mathbb{F}_{p'}$ such that p' is the smallest prime that satisfies $p' > p \cdot \lceil \log n \rceil$. To verify the validity of $C_0(b) = \bigcup_{v \in b} C_0(v)$, we apply a similar idea with respect to the set $D_0(b) = \bigcup_{i \notin F(b)} \{(i, \varphi^b_{i-1}(r'))\}$. Observe that all the multisets that are involved in the equality protocols (and thus, the degrees of all polynomials) are of size $O(\log^2 n)$.

Correctness for outer-block edges. The completeness follows directly from the definition of the distinguishing index and the completeness of the multiset equality protocol. Regarding soundness, suppose that for some edge (u,v) directed from u to v, it holds that $v \prec u$. Denote by (i,j) the pair assigned to the edge (u,v) by the prover in the commitment scheme.

First, consider the case that u and v are in the same block b (but (u,v) is labeled as an outer-block edge by the prover). Notice that (i,j) can be in at most one of the sets $D_0(b), D_1(b)$. This is because $i \in F(b)$ implies $(i,j) \notin D_0(b)$ and $i \notin F(b)$ implies $(i,j) \notin D_1(b)$. Assume w.l.o.g. that $(i,j) \notin D_0(b)$. Notice that by construction $(i,j) \in C_0(b)$, which means that the compared multisets cannot be equal. Hence, by the soundness of the multiset equality protocol, the verifier rejects in this case with probability $1 - \log^2 n/(p \log n) = 1 - 1/\text{poly} \log n$.

Now, suppose that u and v are in different blocks $b_u \neq b_v$. Notice that by construction, $(i,j) \in C_0(b_u)$ and $(i,j) \in C_1(b_v)$. If $i \in F(b_u)$ or $i \notin F(b_v)$, then the soundness follows from a similar argument to the former case. Otherwise, by the definition of the distinguishing index and by the soundness of the multiset equality protocol, it follows that $\varphi^{b_u}_{i-1}(r') \neq \varphi^{b_v}_{i-1}(r')$ with probability $1 - 1/\text{poly} \log n$. If this is the case, then it must be that either $j \neq \varphi^{b_u}_{i-1}(r')$ or $j \neq \varphi^{b_v}_{i-1}(r')$. Let us condition on this event and assume w.l.o.g. that $j \neq \varphi^{b_u}_{i-1}(r')$. Then, $(i,j) \notin D_0(b_u)$ and since $(i,j) \in C_0(b_u)$, the soundness of the multiset equality protocol suggests that the verifier rejects with probability $1 - 1/\text{poly} \log n$.

References

1. Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 2426–2458. SIAM, 2022. doi:10.1137/1.9781611977073.97.
2. Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. Local certification of graph decompositions and applications to minor-free classes. *J. Parallel Distributed Comput.*, 193:104954, 2024. doi:10.1016/J.JPDC.2024.104954.
3. Nicolas Bousquet, Laurent Feuilloley, and Sébastien Zeitoun. Local certification of local properties: Tight bounds, trade-offs and new parameters. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 21:1–21:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.21.
4. Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-offs in distributed interactive proofs. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.DISC.2019.13.
5. Louis Esperet and Benjamin Lévêque. Local certification of graphs on surfaces. *Theor. Comput. Sci.*, 909:68–75, 2022. doi:10.1016/J.TCS.2022.01.023.
6. Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Compact distributed certification of planar graphs. *Algorithmica*, 83(7):2215–2244, 2021. doi:10.1007/S00453-021-00823-W.
7. Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Local certification of graphs with bounded genus. *Discret. Appl. Math.*, 325:9–36, 2023. doi:10.1016/J.DAM.2022.10.004.
8. Pierre Fraigniaud, François Le Gall, Harumichi Nishimura, and Ami Paz. Distributed quantum proofs for replicated data. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ITCS.2021.28.
9. Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2010. doi:10.1016/J.JCSS.2009.07.002.
10. Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST computation with short advice. *Theory Comput. Syst.*, 47(4):920–933, 2010. doi:10.1007/S00224-010-9280-9.
11. Pierre Fraigniaud, Pedro Montealegre, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. On distributed merlin-arthur decision protocols. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2019. doi:10.1007/978-3-030-24922-9_16.
12. François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura. Distributed quantum interactive proofs. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 42:1–42:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.42.
13. Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks I: planar embedding. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 29–38. ACM, 2016. doi:10.1145/2933057.2933109.
14. Yuval Gil and Merav Parter. New distributed interactive proofs for planarity: A matter of left and right, 2025. doi:10.48550/arXiv.2505.00338.

15 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

16 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory Comput.*, 12(1):1–33, 2016. doi:10.4086/TOC.2016.V012A019.

17 Atsuya Hasegawa, Srijita Kundu, and Harumichi Nishimura. On the power of quantum distributed proofs. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 220–230. ACM, 2024. doi:10.1145/3662158.3662788.

18 John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

19 Gillat Kol, Rotem Oshman, and Raghuvansh R Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 255–264, 2018. URL: https://dl.acm.org/citation.cfm?id=3212771.

20 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Comput.*, 22(4):215–233, 2010. doi:10.1007/S00446-010-0095-3.

21 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1096–115. SIAM, 2020. doi:10.1137/1.9781611975994.67.

Appendix

A Complexity of the LR-Sorting Protocol

For ease of presentation, our protocol is described in separate stages. Here, we observe that parts of the stages can be parallelized. First, we observe that the block construction stage can be implemented in three interaction rounds. Indeed, it starts with the prover encoding the block positions along with a proof that each block receives two consecutive numbers. Then, the verifier interacts with the prover to compute two multiset equality polynomials within each block. This can be done in two additional interaction rounds for a total of three. Similarly, the proofs of $u \prec v$ for inner-block edges (u, v), and the commitment scheme of outer-block edges can be completed within three rounds. Moreover, a correct execution of these steps does not depend on the execution of the block construction, thus they can be executed in parallel. We also note that the multiplicity values M_v that are presented in the verification stage of outer-block edges can actually be precomputed by the prover and assigned during the first interaction (they are placed in the verification scheme strictly for the sake of clear presentation). Therefore, after three interaction rounds, it is the verifier's turn to speak and the remaining task is the multiset equality protocol of the verification scheme of outer-block edges (here, notice that the verification scheme cannot be executed sooner as it depends on the values assigned in the commitment scheme). This takes two additional interaction rounds for a total of five rounds. Regarding proof size, a bound of $O(\log \log n)$ is straightforward from the construction.

B Path-outerplanarity

In this section, we present a protocol that uses LR-sorting as a sub-task to decide whether a given graph is path-outerplanar. The properties of the protocol are specified in the following lemma.

▶ **Lemma B.1.** *Suppose that there exists a distributed interactive proof for LR-sorting verification in planar graphs running in t interaction rounds. Let ℓ be the proof size, ϵ_c be the completeness error, and ϵ_s be the soundness error of the LR-sorting protocol. Then, there is a distributed interactive proof for path-outerplanarity running in $\max\{t, 3\}$ rounds and admitting a proof size of $O(\ell)$, a completeness error of ϵ_c, and a soundness error of $\epsilon_s + 2^{-\ell}$.*

As a consequence, we get Theorem 1.2.

Proof of Theorem 1.2. The protocol is obtained by plugging the LR-sorting protocol of Lemma 4.2 into the statement of Lemma B.1. ◀

The rest of the section is dedicated to the description of the protocol that proves Lemma B.1. For clarity, the protocol is described in separate stages without regard for the number of interaction rounds. Then, by the end of the section, we explain how the protocol can be implemented within the desired amount of interaction. Throughout, $c > 0$ is defined as a positive constant that can be made large enough to support the protocol's soundness guarantee.

Committing to a path. The protocol starts by having the prover commit to a Hamiltonian path P of G. To encode P, the prover uses the labels of Lemma 2.3 where P is rooted at the leftmost node in the path. Each node can verify that it has at most one child in the given tree encoding. Additionally, to verify that the given subgraph is indeed a Hamiltonian path of the graph, the prover and verifier execute the protocol of Lemma 2.5 amplified by means of a $c \cdot \ell$ parallel repetition.

Observe that if the graph is indeed path-outerplanar, then the prover can successfully send the verifier a Hamiltonian path. Consequently, each node knows its path-edges and is able to differentiate between its right and left neighbor on the path. On the other hand, if the graph is not path-outerplanar, then the graph is either not Hamiltonian or not outerplanar. In the former case, the prover is not able to provide a Hamiltonian path which causes the verifier to reject with probability $1 - 2^{-\Theta(\ell)}$; in the latter case, the prover is able to send the verifier a Hamiltonian path but the non-path edges are not properly nested.

LR-sorting. This stage starts by having the prover inform the verifier whether $u \prec v$ or $v \prec u$ for every edge $(u, v) \in E$. To see how this is achieved, recall that in the simulation of edge-labels that proves Lemma 2.4, e's label is written within the label of one of its endpoints. Let us refer to that endpoint as the endpoint *accountable* for e. So, if u is accountable for e, then the prover assigns the bit 1 to u's sub-label associated with e to signify that $u \prec v$, and 0 otherwise.

Following this assignment, the goal of the verifier is to check that all edges were labeled correctly by the prover, i.e., to check that if an edge (u, v) was labeled $u \prec v$, then indeed u appears before v in P. To that end, the prover and verifier execute an LR-sorting protocol. To create an instance for LR-sorting, the edges of the graph are oriented according to the prover's labeling. That is, if edge (u, v) was labeled $u \prec v$, then it is oriented from u to v.

Notice that if the verifier accepts the LR-sorting instance, then this means that the prover labeled all edges correctly (up to a soundness error of ϵ_s). So, for the rest of the protocol, we assume that for every non-path edge $e = (u, v)$, both endpoints know whether $u \prec v$ or $v \prec u$. Notice that in particular, this means that each $v \in V$ can distinguish between its left and right edges.

Nesting verification. In this final stage, the goal is to verify that the non-path edges are properly nested. The stage starts with the prover informing the endpoints of each non-path edge $e = (u, v)$, $u \prec v$, whether it is the longest u-right edge and whether it is the longest v-left edge. This is done by assigning two bits within the label of the endpoint accountable for e similarly to the previous stage.

Upon receiving the edge-labels, the verifier at each node $v \in V$ runs the following checks. If v has any right (resp., left) edges, then the verifier checks that exactly one of them is marked as longest v-right (resp., v-left) edge. In addition, for every right (resp., left) edge (v, u) that was not marked longest v-right (resp., v-left), the verifier checks that it was marked longest u-left (resp., u-right). If one of the checks fail, then the verifier immediately rejects. Otherwise, v samples a bitstring $s_v \in \{0,1\}^{c\cdot\ell}$ uniformly at random and sends it to the prover. For each non-path edge (u, v) such that $u \prec v$, define its *name* to be the pair (s_u, s_v).

After receiving the s_v values from all nodes, the prover assigns to each edge e its name through a sub-label $\texttt{name}(e)$ and its successor's name through a sub-label $\texttt{succ}(e)$ where the name of the virtual edge $e^* = (u^*, v^*)$ is defined by the designated symbol $\perp$ (recall that e^* is the successor of edges with no real successor in the graph). Additionally, if edge $e = (u, v)$ has predecessors $(u_1, v_1), \ldots, (u_k, v_k)$ such that $u \preceq u_1 \prec v_1 \preceq \cdots \preceq u_k \prec v_k \preceq v$, then the prover assigns the label $\texttt{above}(w) = \texttt{name}(e) = (s_u, s_v)$ to every node w such that $(u \prec w \preceq u_1) \vee (v_1 \preceq w \preceq u_2) \vee \cdots \vee (v_k \preceq w \prec v)$. In other words, the prover assigns e's name to all nodes for which e is the first edge drawn entirely above them (including the endpoints of e's predecessors; excluding the endpoints of e). In particular, if $e = (u, v)$ does not have any predecessors, then $\texttt{above}(w) = \texttt{name}(e)$ for all nodes w such that $u \prec w \prec v$. Observe that by definition, each node is associated with only one such edge and thus, receives only one edge name.

Consider a label assignment to the nodes and non-path edges. First, for each non-path edge e, its endpoints verify that $\texttt{name}(e)$ is consistent with their sampled values. Then, each node $v \in V$ checks that there exists an ordering $e_1^+, \ldots, e_k^+$ of its right edges, and an ordering $e_1^-, \ldots, e_{k'}^-$ of its left edges such that the following conditions are satisfied:

1. e_k^+ and $e_{k'}^-$ are marked as the longest v-right and v-left edges, respectively.
2. $\texttt{succ}(e_i^+) = \texttt{name}(e_{i+1}^+)$ for all $1 \le i < k$, and $\texttt{succ}(e_i^-) = \texttt{name}(e_{i+1}^-)$ for all $1 \le i < k'$.
3. $\texttt{above}(v) = \texttt{succ}(e_k^+) = \texttt{succ}(e_{k'}^-)$.
4. if u is v's right neighbor on the path, then $\texttt{name}(e_1^+) = \texttt{above}(u)$ if the set of v's right edges is non-empty, and $\texttt{above}(v) = \texttt{above}(u)$ otherwise.
5. if u is v's left neighbor on the path, then $\texttt{name}(e_1^-) = \texttt{above}(u)$ if the set of v's left edges is non-empty, and $\texttt{above}(v) = \texttt{above}(u)$ otherwise.

We note that a pair of orderings that satisfies the described conditions does not have to be unique. Also, notice that nodes which are not incident on any non-path edges only need to check that they were assigned the same value as their neighbors on the path (conditions (4) and (5)). This concludes the description of the nesting verification. We go on to establish its correctness.

Correctness of nesting verification. Towards proving the completeness and soundness of the nesting verification, we show the following two observations.

▶ **Observation B.2.** *Fix some node $u \in V$. If the prover marks the longest u-right or the longest u-left edge incorrectly, then the verifier rejects the instance with probability $1 - 2^{-c\cdot\ell}$.*

Proof. Suppose that edge (u, v) is the longest u-right edge but not marked as such. Recall that by the initial verification conditions, (u, v) must be marked as the longest v-left edge (otherwise the verifier rejects). If v has a right edge, then by verification conditions (1)

and (3), the value $\texttt{succ}(u,v)$ should be identical to the value $\texttt{succ}(v,w)$, where (v,w) is the right edge of v which is marked as longest. If v does not have a right edge, then by verification conditions (3) and (4), the value $\texttt{succ}(u,v)$ should be identical to the value the value $\texttt{above}(w')$ where w' is v's right neighbor on the path. In either case, following the verification conditions we get that $\texttt{succ}(u,v)$ should be identical to $\texttt{name}(u',v')$ of some edge (u',v') such that $v \prec v'$. Moreover, the edge (u',v') is fully determined by the marking of longest left and right edges by the prover (and in particular, determined before the sampling of names). Note that since (u,v) is the longest u-right edge and $v \prec v'$, it must hold that $(u,v') \notin E$ and thus, $u' \neq u$. On the other hand, since (u,v) is not marked as the longest u-right edge, by condition (2), the first element of $\texttt{succ}(u,v)$ should be s_u. So, the verifier rejects unless $s_u = s_{u'}$ which happens with probability $2^{-c\cdot\ell}$. The case of longest left edges follows a similar reasoning. ◀

Going forward with the correctness proof, we shall assume that all longest left/right edges are marked correctly. For two nodes $u \prec v$, denote by $P_{u,v}$ the set of nodes on the (u,v)-subpath in G.

▶ **Observation B.3.** *Suppose that for a non-path edge (u,v), it holds that $G(P_{u,v})$ is path-outerplanar w.r.t. $P_{u,v}$ (i.e., the edges of $G(P_{u,v})$ are properly nested within $P_{u,v}$). If the verifier accepts the instance, then $\texttt{succ}(u',v')$ is the name of the successor of (u',v') in $G(P_{u,v})$ for all non-path edges $(u',v') \neq (u,v)$, $u \preceq u' \prec v' \preceq v$.*

Proof. Let (x,y) be a non-path edge in $G(P_{u,v})$ and let (x_ℓ,y_ℓ) and (x_r,y_r) be its leftmost and rightmost predecessors, respectively. First, if $y_r = y$, then it must hold that $x \prec x_r$ which means that $(x_r,y_r) = (x_r,y)$ is not the longest y-left edge. Therefore, by condition (2) it must hold that the second element of $\texttt{succ}(x_r,y_r)$ is s_y. Now, suppose that $y_r \prec y$. Here, since (x_r,y_r) is a predecessor of (x,y), it follows that (x_r,y_r) is the longest y_r-left edge. Applying conditions (1), (3), and (4) along the (y_r,y)-path, we once again get that the second element of $\texttt{succ}(x_r,y_r)$ must be s_y. For similar reasoning, we can deduce that the first element of $\texttt{succ}(x_\ell,y_\ell)$ is s_x. Now, we observe that by conditions (1), (3), (4), and (5), every pair of adjacent siblings must have the same $\texttt{succ}(\cdot)$ field. Therefore, every predecessor (x',y') of (x,y) must satisfy $\texttt{succ}(x',y') = (s_x,s_y) = \texttt{name}(x,y)$ which concludes our proof. ◀

We can now prove the completeness and soundness of our protocol.

▶ **Lemma B.4.** *The described nesting verification admits perfect completeness and a soundness error of $2^{-\Theta(\ell)}$.*

Proof. We start from completeness. First, we note that by Observation 2.1, the honest prover can mark each edge (u,v) as longest u-right/v-left correctly. Furthermore, observe that the feasibility of the $\texttt{name}(\cdot)$, $\texttt{succ}(\cdot)$, and $\texttt{above}(\cdot)$ labels assigned by the honest prover is guaranteed by Observation 2.2. Now, consider some node $v \in V$ and let $e^-_{k'} = (v,u^-_{k'}),\ldots,e^-_1 = (v,u^-_1), e^+_1 = (v,u^+_1),\ldots,e^+_k = (v,u^+_k)$ be its incident non-path edges such that $u^-_{k'} \prec \cdots \prec u^-_1 \prec v \prec u^+_1 \prec \cdots \prec u^+_k$. Given the labels assigned by the honest prover, the orderings $e^-_1,\ldots,e^-_{k'}$ and $e^+_1,\ldots,e^+_k$ defined on the left and right edges of v satisfy all the verification conditions, thus causing the verifier to accept.

We now establish the soundness guarantee. Let us define (u,v) as an edge that admits a crossing edge (u',v') such that $u \prec u' \prec v \prec v'$ but not a crossing edge (u',v') such that $u' \prec u \prec v' \prec v$. That is, (u,v) is not crossed by edges that has an endpoint to the left of u. Moreover, assume that (u,v) is the deepest nested such edge, i.e., every edge $(x,y) \neq (u,v)$

where $u \preceq x \prec y \preceq v$ does not admit a crossing edge. Observe that if there exists a pair of crossing edges in the graph, then there exists an edge (u, v) satisfying the assumptions stated above.

We start from the case where (u, v) is the longest v-left edge. Define $u \prec u' \prec v$ to be the rightmost node incident on a right edge that crosses (u, v) and define (u', v') as the longest u'-right edge (by definition, (u', v') crosses (u, v)). Let $e_1 = (u_1, v), e_2 = (u_2, v), \ldots, e_k = (u_k, v)$ be v's left edges ordered such that $u = u_k \prec \cdots \prec u_2 \prec u_1 \prec v$. Observe that by the assumptions on (u, v), it follows that $u' \preceq u_{k-1}$. Moreover, all edges that are drawn below e_{k-1} are properly nested. Therefore, Observation B.3 implies that if the verifier accepts the instance, then $\texttt{succ}(e_i) = \texttt{name}(e_{i+1})$ for every $1 \leq i < k - 1$. Furthermore, recall that (u, v) is marked as the longest v-left edge. Thus, for condition (2) to be satisfied at node v, it must also hold that $\texttt{succ}(e_{k-1}) = \texttt{name}(e_k) = (s_u, s_v)$.

To show that the verifier is likely to reject in this case, the idea is to define a sequence of edges that must agree with e_{k-1} on their $\texttt{succ}(\cdot)$ value, but also must have $s_{u'}$ as their $\texttt{succ}(\cdot)$ value's first element. This implies that the verifier rejects unless $s_u = s_{u'}$ which happens with probability $1/\text{poly}\log n$. The sequence $(x_1, y_1), \ldots, (x_t, y_t)$ of edges is defined as follows. Start by taking $x_1 \preceq u_{k-1}$ to be the closest node to u_{k-1} incident on a left edge and set (x_1, y_1) as the longest x_1-left edge. Then, take $x_2 \preceq y_1$ to be the closest node to y_1 incident on a left edge and set (x_2, y_2) as the longest x_2-left edge. Continue this process until reaching y_t such that all nodes w such that $u' \prec w \preceq y_t$ are not incident on a left edge. Notice that the sequence construction is feasible since by our assumption on (u, v), no edge within (u, v) crosses (u', v') (and thus, the sequence is entirely to the right of u'). Moreover, since u' is the rightmost node incident on a right edge crossing (u, v), it follows that every edge (x_i, y_i) in the sequence is the longest y_i-right edge. We note that the verification conditions dictate that every pair of adjacent edges in the sequence should have the same $\texttt{succ}(\cdot)$ value and that $\texttt{succ}(x_1, y_1) = \texttt{succ}(e_{k-1})$. On the other hand, for u' to satisfy condition (4), the first element of $\texttt{succ}(x_t, y_t)$ must be $s_{u'}$ which concludes the soundness for this case.

We move on to the case where (u, v) is not the longest v-left edge. If (u, v) is also not the longest u-right edge, then by the construction the verifier rejects. So, assume that (u, v) is the longest u-right edge. Let $u \prec u' \prec v$ be the leftmost node incident on an edge crossing (u, v) and let (u', v') be the longest u'-right edge (by definition, (u', v') crosses (u, v)). By similar measures to the previous case, it is possible to find a sequence $(x_1, y_1), \ldots, (x_t, y_t)$ of edges such that must satisfy $\texttt{succ}(x_1, y_1) = \cdots = \texttt{succ}(x_t, y_t) = \texttt{succ}(u', v')$; and the first element of $\texttt{succ}(x_i, y_i)$ is s_u for each $1 \leq i \leq t$. On the other hand, by similar reasoning to the one presented in the proof of Observation B.2, it must hold that $\texttt{succ}(u', v') = \texttt{name}(w, w')$ for some edge (w, w') such that $v' \preceq w'$. Moreover, this edge is fully determined by the marking of longest left and right edges by the prover (and in particular before the sampling of names). Recall that $v \prec v' \preceq w'$ and that (u, v) is the longest u-right edge. Therefore, it must hold that $w \neq u$ which means that the probability of $s_u = s_w$ is at most $2^{-c \cdot \ell}$. ◀

Analysis of the protocol. By construction, the proof size of the protocol is $O(\ell)$. Moreover, all stages apart from the black-box use of the LR-sorting protocol admit perfect completeness and a soundness error of $2^{-\Theta(\ell)}$. Thus, by union bound arguments, the completeness error of the protocol is ϵ_c and the soundness error is $\epsilon_s + 2^{-\Theta(\ell)}$. Of course, taking a sufficiently large c, we can have a soundness error of $\epsilon_s + 2^{-\ell}$ as desired. Finally, regarding the number of interaction rounds, we note that all stages can be executed in parallel without affecting the correctness of the algorithm. It is straightforward to see that the stages committing to a path and nesting can be implemented in 3 interaction rounds. Since the LR-sorting protocol requires t rounds, we get that in total the protocol runs in $\max\{t, 3\}$ rounds.

C Additional Related Work

Beyond planarity. Following the introduction of efficient distributed proof systems for planarity [21, 6], researchers have become interested in distributed proof systems for other graph families. The aforementioned compiler of [21] implies a three-round distributed interactive protocol with $O(\log n)$ proof size for families of sparse graphs (i.e., $m = O(n)$ edges) that admit a linear-time recognition algorithm. These include, e.g., *bounded genus* graphs and *outerplanar* graphs. Distributed proofs for bounded genus graphs were studied further in [5, 7] where proof labeling schemes with a proof size of $O(\log n)$ are presented. For outerplanar graphs, a proof labeling scheme with a proof size of $O(\log n)$ is presented in [2]. Additionally, the authors show similar results for a myriad of minor-free graphs.

Distributed interactive proofs variants. In [4], trade-offs between different parameters of the DIP model are explored. The parameters considered include the form of randomness, the complexity measures, and the number of interaction rounds. Recently, the notion of *distributed quantum interactive proofs* was introduces by the authors of [12] as a quantum variant of distributed interactive proofs. The main result of [12] is a generic transformation from a k-round "standard" proof into a 5-round quantum proof for any constant $k > 5$. Distributed quantum proofs have also been considered in a non-interactive setting in [8, 17]. Another exciting variant that was introduced recently in [1] is that of a *distributed zero-knowledge proof*. In particular, the authors adapt the classical notion of knowledge from the centralized setting (as defined in [15]) to a distributed setting.

PIPQ: Strict Insert-Optimized Concurrent Priority Queue

Olivia Grimes
Lehigh University, Bethlehem, PA, USA

Ahmed Hassan
Lehigh University, Bethlehem, PA, USA

Panagiota Fatourou
FORTH ICS, Heraklion, Greece
Department of Computer Science, University of Crete, Heraklion, Greece

Roberto Palmieri
Lehigh University, Bethlehem, PA, USA

Abstract

This paper presents PIPQ, a strict and linearizable concurrent priority queue whose design differs from existing solutions in literature because it focuses on enabling parallelism of insert operations as opposed to accelerating delete-min operations, as traditionally done. In a nutshell, PIPQ's structure includes two levels: the worker level and the leader level. The worker level provides per-thread data structures enabling fast and parallel insertions. The leader level contains the highest priority elements in the priority queue and can thus serve delete-min operations. Our evaluation, which includes an exploration of different data access patterns, operation mixes, runtime settings, and an integration into a graph-based application, shows that PIPQ outperforms competitors in a variety of cases, especially with insert-dominant workloads.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Priority Queue, Concurrent Data Structures, Synchronization

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.35

Related Version *Full Version*: `https://arxiv.org/abs/2508.16023`

Supplementary Material *Software (Source Code)*: `https://github.com/sss-lehigh/pipq` [12]

Funding This material is based upon work supported by the National Science Foundation under Grant No. CNS-2045976 and by the Greek Ministry of Education, Religious Affairs and Sports through the HARSH project (project no. ΥΠ3ΤΑ - 0560901), which is carried out within the framework of the National Recovery and Resilience Plan "Greece 2.0" with funding from the European Union - NextGenerationEU.

Acknowledgements The authors would like to acknowledge Mike Spear for all his input, feedback, and support.

1 Introduction

Concurrent data structures are at the heart of modern applications and thus, their efficient and scalable implementation is of great importance. Priority queues, particularly, are widely used data structures with many applications in diverse domains [5, 24, 20, 21, 22, 32, 3, 13, 6]. Applications include graph and encoding algorithms, big data analysis, task scheduling and load balancing, event-driven simulation, query optimization, databases (e.g., in transaction management, deadlock handling, process handling), network routing, and more.

A priority queue supports two operations: *insert*, which inserts a key-value pair in the priority queue, and *delete-min*, which deletes and returns the element with the smallest key (indicating the *highest priority*) in the queue.


© Olivia Grimes, Ahmed Hassan, Panagiota Fatourou, and Roberto Palmieri;
licensed under Creative Commons License CC-BY 4.0



The last decade has been characterized by innovations in the design of priority queues focusing mainly on improving the performance of delete-min [18, 1, 30, 26, 24, 33], which is unquestionably a sequential bottleneck for most designs, since these operations all target the same highest priority element. These innovations focus mostly on relaxing the semantics of the priority queue itself to allow for more parallelism when serving the delete-min operation. In relaxed designs, the rank of the removed element is within some defined bound of the highest priority element, although it may not be the highest priority element itself. However, relaxed semantics are not sufficient for many applications in which the single highest priority element must be removed each time. Some examples that may require strict semantics include risk management systems to ensure critical tasks are addressed first, real-time systems such as medical systems and devices, which must optimally treat patients, and in security applications in which high-priority messages or threats must be processed ahead of all other requests.

Despite the highlighted importance of such strict behavior of priority queues in many applications, the focus on relaxed priority queues in the last decade left a clear gap in literature and made the performance of state-of-the-art strict priority queues significantly inferior to their relaxed counterparts. However, if relaxation is not a viable option, the performance of applications needing strict semantics will always be limited by the underlying priority queue. In this paper, we fill this gap as follows. Optimizing on delete-min in the strict setting is difficult due to its sequential nature. Thus, while still aiming to alleviate the performance impact of the pessimistic nature of delete-min, we shift our focus to maximizing the performance and parallelism of insert operations. In fact, achieving a significant speedup for insert operations can offset the overhead of the delete-min operations. Beyond that, there is a class of applications for which the performance of insert operations is important. Examples include graph analytics [17], data series analysis [5, 20, 21, 22], and computation over streamed data [23].

The core intuition is to improve the performance of insert operations by using a set of data structures, one per thread, to avoid synchronization with other threads when the inserted element is not among those with the highest priority. When this condition is met, which we claim to be the common case for many applications [17, 5, 20, 21, 22, 23], insertions can effectively be embarrassingly parallel. This ensures perfect scalability, Non-Uniform Memory Access (NUMA)-awareness, and near-optimal performance.

We developed this intuition into PIPQ[1], our linearizable, concurrent priority queue whose main strength is its high-performance insert operation. PIPQ's design encompasses a hierarchical structure:

- In the first level, called the *worker level*, we deploy one min-heap [16] per thread. Insertions are performed to the thread's min-heap unless the thread has observed the highest priority element to be inserted.
- The second level, called the *leader level*, collects the highest-priority elements in PIPQ, hence creating a set of candidate elements to be returned by delete-min.

Delete operations remove elements from the leader level, which contains the most minimal element(s) inserted by each thread, and thus the most minimal element(s) in the entire structure. To avoid degrading the performance of delete-min due to this hierarchical structure, we employ the combining technique [15, 9] to minimize contention between concurrent removals.

[1] Pronounced "Pip Q", standing for Parallel Inserts Priority Queue.

We develop PIPQ in C++[2] and compare its performance against two state-of-the-art strict linearizable priority queues, namely the Lindén-Jonsson priority queue [18], and the Lotan-Shavit priority queue [28]. Experiments are conducted on an Intel server equipped with 96 cores over four processors. Applications include a microbenchmark, in which threads execute a mix of workloads, and benchmarks that mimic the access patterns generated by recent applications of priority queues, in which inserting is a dominating factor. Evaluation of the widely used single-source shortest path (SSSP) algorithm is additionally included in the extended version of the paper. Across experiments, our evaluation reveals that the proposed algorithm exhibits great performance benefits compared to competitors when insertions dominate the workload. Importantly, PIPQ does not sacrifice performance in workloads exhibiting an equal split of both operations, nor in delete-min heavy workloads.

2 Related Work

Lotan and Shavit [28] were the first to use a Skiplist [25] as a concurrent priority queue, a design which has since largely become the standard for implementing concurrent priority queues. Notably, the Lotan and Shavit [28] concurrent priority queue is not linearizable, instead achieving the weaker condition of being quiescently consistent [16]. Sundell and Tsigas [29] created a lock-free, linearizable concurrent priority queue based on a skip-list. Lindén and Jonsson then presented their strict, linearizable priority queue algorithm [18]; their enhancement of Lotan and Shavit's priority queue to the delete-min operation delays physical deletion of elements to improve performance. The evaluation provided by Lindén and Jonsson shows that their priority queue consistently outperforms that of Sundell and Tsigas for various types of experiments. Braginsky et al. [4] similarly use a skiplist-based design, modifying data nodes to be lock-free chunks and leveraging elimination to speed up concurrent delete-mins, and inserts that insert to the first chunk. While PIPQ is optimized to increase the parallelism of insert operations, the work in [4] still uses a non-thread-local structure but reduces synchronization through coalescing elements. Zhang and Dechev [31] implement a lock-free priority queue based on a multi-dimensional list. Their implementation is only quiescently consistent. Similar to these solutions, PIPQ aims to create a strict linearizable priority queue, but with a new design which does not include use of a skip-list and aims to optimize insertions, all while achieving linearizability.

More recent work on priority queues has focused on relaxed solutions. In these designs, the element removed by a delete-min may not be the highest priority element itself, but is within some defined bound of it. Spraylist [1] is a well-known relaxed priority queue, and many other relaxed solutions have been proposed [30, 33, 27, 26, 24]. Unlike these relaxed solutions that optimize for the delete-min operation, PIPQ provides a strict priority queue that optimizes for insert.

A recent related work of interest is the Stealing Multi-Queue (SMQ) priority scheduler by Postnikova et al. [24], which innovates on the prior Multi-Queues work [26]. SMQ uses a design similar to that of the worker level of PIPQ, such that it maintains per-thread heaps. As a result, insert operations are optimized, and, in addition to that, it relaxes the semantics of the delete-min operation with the use of the so-called stealing buffers. Despite their similarities, removing the relaxation of SMQ would require additional synchronization, which is effectively the goal accomplished by PIPQ.

[2] The source code for PIPQ is publicly available here: `https://github.com/sss-lehigh/pipq`

3 Overview of PIPQ

PIPQ is a linearizable priority queue that provides the traditional priority queue API: `Insert`, which inserts some key-value pair, and `DeleteMin`, which removes the element with the smallest (i.e., highest priority) key in the structure. Our priority queue supports duplicate keys and/or values. Figure 1 illustrates PIPQ's architecture, including an example instantiation relevant to a specific thread (red / bolded).

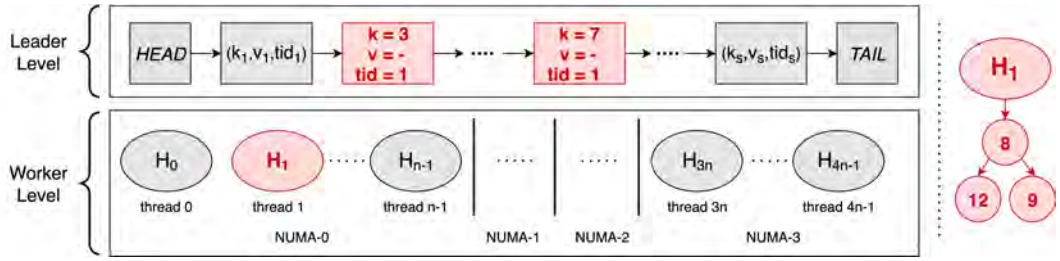


Figure 1 PIPQ on a 4-NUMA node testbed. Each oval represents a per-thread worker-level min-heap; "n" represents the number of threads per NUMA node. Squares represent nodes of the leader-level linked list; "s" represents the size of the leader-level linked list.

To create highly parallel insertions, PIPQ maintains a two-level hierarchy of supporting data structures. The two levels include the worker level and the leader level. The worker level provides fast, parallel insertions with minimal synchronization because each thread p inserts to its own worker-level heap, H_p. Each H_p is implemented as a conventional min-heap protected by a single global lock. The simplicity of this level's design is justified by the fact that most of the time no other threads need access to H_p, as subsequently discussed. At the leader level, the algorithm maintains a sorted linked list L of keys, which stores the highest priority elements in PIPQ, and is meant to serve delete-min requests.

Elements may be inserted to the worker or leader level and may move between levels in order to maintain the following invariant: *for each thread p, every element in L tagged with p (that is, p inserted the element into the global data structure) has greater or equal priority than the highest priority element in the worker-level heap of p, H_p.* This ensures that when we remove the highest priority element from the leader level, it is, in fact, the highest priority element in the priority queue, allowing us to achieve linearizability without relaxing operation semantics. The items in red in Figure 1 provide an example instantiation of PIPQ relative to a specific thread, denoted thread 1. The highest priority element in H_1 has priority 8. The corresponding nodes inserted by thread 1 in L have a higher priority (smaller value of the key, `k`) than 8, and are ordered in L by `k`.

A thread p invoking insert on PIPQ will insert to the leader level only when the new element is of higher priority than the highest priority element in H_p (or when H_p is empty). Otherwise, the element will be inserted into H_p. The leader level is intentionally kept small (explained below), so we expect most insertions to occur at the worker level.

The delete-min operation leverages the combining technique [15, 9] as used in several prior works [8, 10, 7, 19]. Electing one thread to perform a sequence of delete-min operations, which are inherently sequential, avoids the costly contention of many threads attempting to access and remove the highest priority element in the leader-level linked list. We call this thread the *coordinator*. A thread p that performs a delete-min operation on PIPQ first announces its operation and then attempts to become the *leader* of its Non-Uniform Memory Access (NUMA) node. Leaders of each NUMA node compete to become the coordinator;

thus, there are up to n leaders competing for the coordinator role at any given time, where n is the number of NUMA nodes in the machine. The coordinator is responsible for serving the delete-min operations of threads that execute on its NUMA node. Thus, either p will become the coordinator and complete its own request along with the other requests of its NUMA node, or its request will be completed by some other thread of its NUMA node that won the race to become the coordinator.

Depending on the sequence of insert and delete-min operations, the leader-level linked list L may grow and shrink arbitrarily. However, the size of L is an important factor in the overall performance of PIPQ. In fact, having many elements in L hinders the performance of insertions for two reasons. First, traversals on L involve many elements. Second, a large size of L increases the likelihood that an insertion has to act at the leader level as opposed to at the worker level, since the range of the priority of elements in L increases with its size. For these reasons, we limit the size of L by requiring insertions to move elements from L down to its worker-level heap if the number of elements that the thread has inserted in L exceeds a certain threshold, which we call `CNTR_MAX`.

On the other hand (and less intuitively), having too few elements in L hinders the performance of delete-min operations. To successfully linearize delete-min operations while ensuring the properties that L needs to satisfy, each thread has to have at least two of the elements it has inserted in L; the justification for requiring at least two elements is detailed in Section 5.1. If this requirement is not met, the coordinator must promote an element from the thread's worker-level heap. Having few elements in L means that the coordinator would need to perform such a promotion more frequently. We address this issue by defining a second threshold, `CNTR_MIN` (with a value less than that of `CNTR_MAX`), and use it to introduce a helping mechanism.

The idea behind helping is that a thread can do work that does not directly impact the operation it is performing but can "help" other operations complete their work in a more efficient manner. Since threads performing the delete-min operation must wait on the coordinator, we initiate helping during this waiting period to reduce the work required by the coordinator. Specifically, if the number of elements inserted by some thread p in L is less than `CNTR_MIN`, p will promote an element from H_p up to the leader level instead of waiting for the coordinator to eventually do so.

We maintain an array of atomic counters containing one slot per thread such that each thread p's slot in the counters array stores the number of elements currently in L that were inserted by p to the global structure. We refer to p's counter value as L_count_p. The use of helping and the gap between `CNTR_MIN` and `CNTR_MAX` is meant to achieve the following two goals: (1) it is less costly for a waiting thread p to help by moving elements from H_p to L, compared to the coordinator needing to do so; thus, we reduce the frequency with which the coordinator must do additional work, and (2) by helping only when L_count_p is less than the minimum threshold (which is, in turn, less than the maximum threshold), we reduce the frequency with which an insertion has to move an element down from the leader level to the worker level.

Before discussing the details of PIPQ, it is important to highlight that the leader-level linked list, L, differs from traditional linked lists in that it requires tracking which thread inserts each element in order to move them to the corresponding worker-level min-heap when necessary. To do so, L stores tuples $(\texttt{key,val,tid}_\texttt{p})$, where $\texttt{tid}_\texttt{p}$ represents the unique identifier of the thread, p, that first inserted this key-value pair in the priority queue. L is sorted based on the key and supports the following operations: *a)* L-Insert(key, val, tid_p), which inserts the tuple $(\texttt{key,val,tid}_\texttt{p})$ in L; *b)* L-DeleteMin, which deletes and returns the

element with the smallest (highest priority) key in L; and *c)* L-DeleteMaxP(lead_largest, $\texttt{tid}_\texttt{p}$), which removes the element with the largest (least priority) `key` that was inserted to PIPQ by the thread with identifier $\texttt{tid}_\texttt{p}$ (to be subsequently inserted into H_p).

4 Operational Workflow

Figure 2 contains flow diagrams for the `DeleteMin` and `Insert` APIs of PIPQ. Recall the invariant maintained between levels, such that the priority of each element in L must be greater than or equal to the priority of the highest priority element from their respective worker level heaps. Deciding on which level to insert, as well as the movement of elements between levels, aims to maintain this invariant. Pseudocode for the leader-level APIs is provided in Section 5.1 and their helper (search) functions are provided in Appendix A. The entire pseudocode of PIPQ is provided in the extended version of the paper.

Insertion Flow. We first describe how an insert operation works in PIPQ, taking into consideration the different levels to which a new element might be inserted. The following description corresponds to the diagram on the left in Figure 2. The steps mentioned below refer to the numbered blocks in the diagram.

Let us consider an insert operation performed by thread p, identified by $\texttt{tid}_\texttt{p}$, that inserts key `k` and value `v`. Per Step 1, the thread begins by locking its local, worker-level min-heap, H_p. In Step 2, it checks the minimum (i.e., highest priority) element in H_p's key, $\texttt{k}_\texttt{p}$, and compares it to `k`. This comparison determines whether the insertion will follow the so called *fast path* of the algorithm or one of two slow paths, named the *slower* and *slowest* path. If `k` is greater than or equal to $\texttt{k}_\texttt{p}$ (i.e., has a lesser priority), then the fast path is taken as the insertion is thus able to be performed in a completely parallel manner. Specifically, p inserts the key-value pair, `k` and `v`, into H_p (Step 3), and then unlocks H_p and returns (Step 11).

On the other hand, if `k` is found to be less than the minimum of H_p at Step 2, then it is necessary to compare `k` to the priority of elements in the leader level in order to maintain proper ordering of elements. The comparisons in Steps 4 and 7 determine if the element should be inserted to the leader-level, in which case the thread follows either the slower path (orange, middle dashed box) or the slowest path (red, right-most dashed box), or if the element should be inserted to the worker level via the fast path (following Steps 7 to 3).

Recall that in order to maximize the number of insertions to a thread's local heap without affecting the efficiency of insertions to L, we set a maximum number of elements for L, `CNTR_MAX`. Assuming Step 4 finds that L_count_p is not equal to `CNTR_MAX` (and is thus smaller than it), we follow the slower path. Thus, in Step 5, we insert to L using L-Insert and then perform the atomic fetch-and-add operation on L_count_p (Step 6) to increment it. Finally, we unlock H_p and return in Step 11.

On the contrary, if L_count_p is determined to be equal to `CNTR_MAX` back in Step 4, this indicates that if the new element is inserted to L, then an element must also be moved down to the worker-level. If `k` is the largest element in L tagged with $\texttt{tid}_\texttt{p}$, then it would be wasted work to insert the element into L just to subsequently remove it to be inserted into H_p. To avoid this situation, we maintain a per-thread pointer, $\texttt{lead_largest}_\texttt{p}$, which points to the node containing the largest key in L tagged with some $\texttt{tid}_\texttt{p}$. Step 7 compares `k` to the priority of the node pointed to by $\texttt{lead_largest}_\texttt{p}$ to determine if we can insert the element to the worker level. If `k` is in fact larger or equal in priority to the current largest element (Step 7), then p inserts the key-value pair, `k` and `v`, into H_p (Step 3), and then unlocks H_p and returns (Step 11).

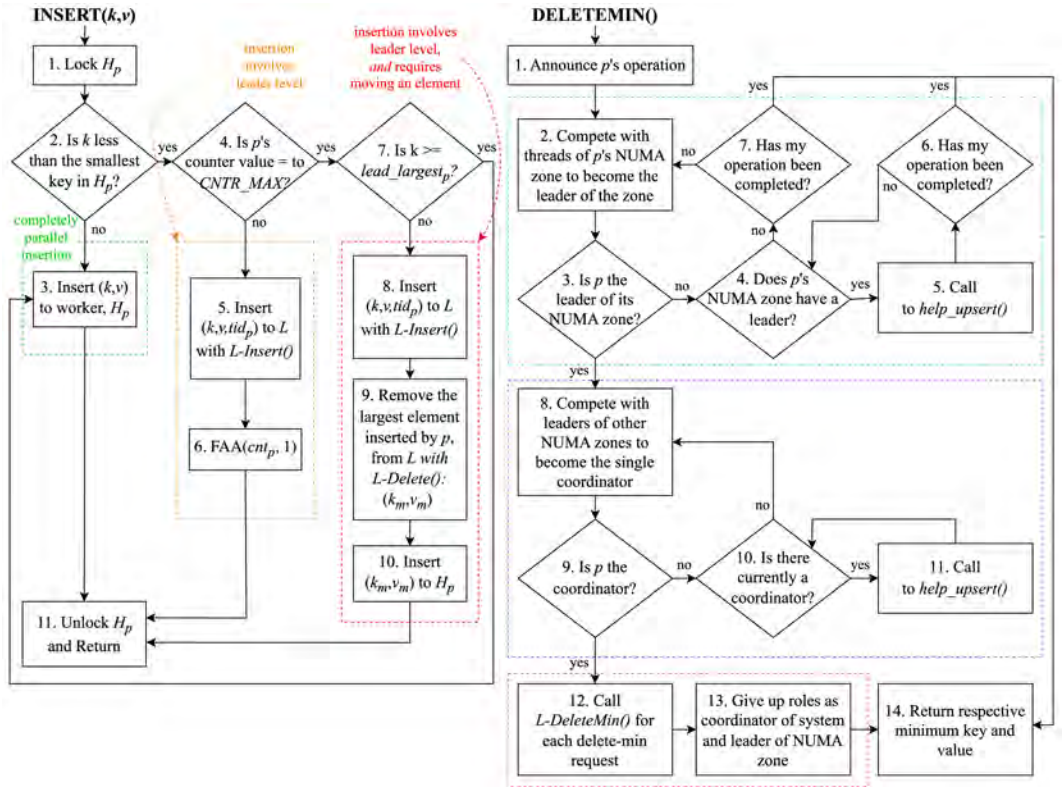


Figure 2 Flow diagrams for the Insert (left) and Delete-Min (right) operations. In each diagram, p represents the thread performing the respective operation.

Otherwise, the new element must be inserted to L, and thus the slowest path is followed. In Step 8, L-INSERT is called to insert tuple ($\texttt{k}$, $\texttt{v}$, $\texttt{tid}_\texttt{p}$) into L. Then in Step 9, L-DELETEMAXP is called to remove the node pointed to by $\texttt{lead_largest}_\texttt{p}$ from L. During the traversal of L, the algorithm tracks the relevant predecessor of $\texttt{lead_largest}_\texttt{p}$ (i.e., the first node tagged with $\texttt{tid}_\texttt{p}$ that appears before the node pointed to by $\texttt{lead_largest}_\texttt{p}$) in order to reset the pointer. Thus, the node pointed to by $\texttt{lead_largest}_\texttt{p}$ is removed from L, $\texttt{lead_largest}_\texttt{p}$ is reset, and L-DELETEMAXP returns the relevant key and value ($\texttt{k}_\texttt{m}$, $\texttt{v}_\texttt{m}$). Finally, ($\texttt{k}_\texttt{m}$, $\texttt{v}_\texttt{m}$) is inserted to H_p in Step 10, and in Step 11 H_p is unlocked and the operation completes.

Delete-Min Flow. We now describe the flow of the delete-min operation. Steps mentioned refer to those in the right diagram of Figure 2.

The delete-min operation begins with thread p announcing its operation in Step 1. The blue (top-most) dashed box in the Figure shows the competition between threads of the same NUMA node to become the leader of their node. Per Steps 3 and 4, if p fails to become the leader, it waits until either its operation is completed by the current coordinator (Step 6) or it eventually succeeds in becoming the leader. While waiting in Steps 4-6, the operation calls a helper routine, HELP-UPSERT, which checks the calling thread's counter value, L_count_p, and compares it to $\texttt{CNTR_MIN}$. If the comparison finds that L_count_p is less than $\texttt{CNTR_MIN}$, thread p will move the highest priority element from H_p to L, updating its counter value accordingly.

If a thread becomes the leader of its NUMA node (Step 8), then it competes to become the coordinator of the entire system, as reflected in the purple (middle) dashed box. If it does not succeed, it follows a nearly identical waiting routine as previously described, the only difference being that it does not check if its operation has been completed since the coordinator only completes operations carried by threads in its own NUMA node.

Once a thread becomes the coordinator (Step 12), it completes the delete-min operations in a combining manner [15, 9]. This is done by calling L-DeleteMin for each entry in the NUMA-local announce array. After each L-DeleteMin call, it atomically decrements L_count_p. Recall that in order to maintain correctness at the leader level, each thread p must have at least two elements in the linked list associated with its identifier, $\texttt{tid}_\texttt{p}$. Thus, after decrementing, if the new value of the counter is less than two, the coordinator must move an element from the relevant worker-level heap up to the leader-level. To do so, the coordinator first gets the lock on the relevant worker-level heap, H_p (unless the coordinator happens to be the thread associated with identifier $\texttt{tid}_\texttt{p}$). It next removes the highest priority from H_p, and then inserts it (augmented with $\texttt{tid}_\texttt{p}$) into L, and finally unlocks H_p if necessary (i.e., if the lock was acquired). Before proceeding to the next active request, the coordinator informs the originating thread that its operation has been completed via the announce array. Once the coordinator has completed all the active operations, it gives up its role as the coordinator and then as the leader of its NUMA node (Step 13) and finally returns its operation (Step 14) to the application.

5 Internal Data Structures Design

5.1 Leader-Level Linked List

The implementation of the leader-level linked list structure, L, combines characteristics from two state-of-the-art lock-free algorithms, namely Harris' sorted singly-linked list algorithm [14] and the Lindén-Jonsson priority queue algorithm [18], which implements the priority queue using a skip list; we borrow ideas from the level-zero list implementation.

The Harris work introduces the idea of marking nodes for removal before performing physical deletion (i.e., unlinking the node) to implement a lock-free linked list. The actual technique used for marking nodes relies on stealing [2] and manipulating the last bit of the pointer[3] pointing to the next node. The physical deletion can then be performed.

An important difference between the Harris algorithm and Lindén-Jonsson's level-zero algorithm lies in their approach to logically marking nodes for removal before performing physical deletion. In fact, in order for the Lindén-Jonsson algorithm to mark a node as logically removed, the preceding node's next pointer is marked instead of the next pointer of the node itself. The algorithm must mark the previous node for deletion in order to maintain a prefix of logically deleted nodes; having this prefix is an optimization that allows for delaying physical removal, which improves performance.

In our leader-level linked list implementation, we use the same optimization as the Lindén-Jonsson algorithm in the L-DeleteMin operation. However, we cannot use this marking approach for the L-DeleteMaxP operation because L-DeleteMaxP removes elements from the middle of the list (as opposed to the front, as in L-DeleteMin). Instead, the L-DeleteMaxP operation follows the approach of Harris' algorithm and marks the next pointer of the node to be deleted. In fact, if we were to mark the previous node's next

[3] Memory is properly aligned to enable the bit-stealing technique.

pointer for removal, it may cause a concurrent insertion to be falsely inserted. Specifically, say a concurrent insertion inserts its new node, n, directly after some node r. It is then possible that n is inserted before the physical removal of r, and upon the physical removal of r, n would also be removed. This situation is prevented by marking r's next pointer since this would prevent the concurrent insertion after r.

These two different marking approaches present a correctness issue: an L-DeleteMin operation and an L-DeleteMaxP operation cannot simultaneously mark the same node. Note that it is only possible for one L-DeleteMin and one L-DeleteMaxP operation on L to interfere at any single time, since at each moment there is only one active thread performing L-DeleteMin (i.e., the coordinator), and only one thread performing L-DeleteMaxP on a specific element, since a thread p only ever moves an element tagged with $\texttt{tid}_p$. To remedy this issue and guarantee that the L-DeleteMaxP and L-DeleteMin operations never interfere, we maintain at least two elements in L from each thread. For this reason, we require that `CNTR_MIN` (and thus, `CNTR_MAX` as well) must be at least two.

Note that we explored other designs for the leader level and found our current implementation to be the most effective. Despite the linear complexity of linked lists, our use of `CNTR_MIN` and `CNTR_MAX`, which reduce the likelihood that either operation must follow its worst-case path, combined with maintaining a small size of the list, makes the complexity not a practical concern. More details are provided in Appendix C.

Implementation Details

The leader-level linked list has two immutable sentinel nodes pointed to by `head` and `tail`, that contain $-\infty$ and $+\infty$, respectively. The actual elements of the list are stored in nodes between the `head` and `tail` nodes. Each node contains a next pointer and the three fields comprising the tuple `key`, `val`, and $\texttt{tid}_p$.

The leader-level linked list (L) supports the following APIs: L-Insert, L-DeleteMaxP, and L-DeleteMin. Internally, L additionally includes helper functions search, searchDelete, and searchPhysDel. These helper functions are all variations of the search function provided by Harris' linked list [14], and more details are included in Appendix A.2 due to space constraints (see Algorithms 4 and 5). The implementation is lock-free, like that of Harris' linked list [14] and Lindén and Jonsson's priority queue [18].

Helper Functions & Marking. Each of the three helper functions, search, searchDelete, and searchPhysDel, must traverse the leader-level linked list by following pointers which may be marked as either *logically deleted*, or *moved*. We steal two bits from each node's next pointer to support the two marking techniques. We denote the bit associated with logical deletion as the DELMIN bit, and the bit associated with moving an element as the MOVING bit. Recall that a node is considered logically deleted if the next pointer of its predecessor node's DELMIN bit is set, and moved if its own next pointer's MOVED bit is set.

The three search functions use the following methods to safely traverse the nodes and set or unset the two bits. Note that there are variants for each function, expressed between the "{}" braces, and the description of the return values for each function discusses them from left to right, respectively.

- **Is_{ LogDel | Moving }_Ref(ptr*)**: returns True/False depending on whether the LOGDEL bit or MOVING bit, respectively, is set in `ptr`.
- **Get_{ LogDel | Moving }_Ref(ptr*)**: returns modified `ptr` with LOGDEL bit set or MOVING bit set, respectively.
- **Get_{ NotLogDel | Unmarked }_Ref(ptr*)**: returns modified `ptr` *without* the LOGDEL bit set, or neither bit set, respectively.

L-Insert. The L-INSERT operation (see Algorithm 1) begins by calling helper function SEARCH (Algorithm 4 in the Appendix), which locates the two nodes to insert between, namely `l_node` and `r_node`. The search function is very similar to that of Harris'; the differences in our implementation account for the different types of marking used by PIPQ.

Once the search returns to L-INSERT, `new_node` is allocated by calling NEWNODE (Line 4) which sets fields `key`, `val`, $\texttt{tid}_\texttt{p}$, and `next`. Finally, a CAS is performed in Line 5 of L-INSERT to swing `l_node`'s next field to point to `new_node`. Upon a successful CAS, the algorithm returns. Otherwise, the operation is attempted again.

L-DeleteMaxP. Recall that the L-DELETEMAXP function removes the last element in L tagged with the calling thread's identifier, to be inserted to the worker level. L-DELETEMAXP begins by storing a copy of `lead_largest` in pointer `start_lead_largest`, to be later used when removing this node from L.

The algorithm then enters a loop and calls helper method SEARCHDELETE (Algorithm 5) in Line 15, which identifies and returns `l_node` and `r_node`, such that `r_node` is the node to be removed from L to move down to the worker-level, and `l_node` is its direct predecessor. The search function additionally resets the thread's `lead_largest` pointer.

Once the left and right nodes are returned to L-DELETEMAXP, a successful CAS in Line 18 indicates that the node has been logically removed from the list and the algorithm breaks from the loop; else it tries again. Lines 21-22 ensure that the node is physically removed from the list; if the CAS in Line 22 fails, the search function SEARCHPHYSDEL is called to ensure physical removal of `r_node`. SEARCHPHYSDEL varies only slightly from SEARCH, such that instead of searching for a certain key value, it searches for `r_node`. If the algorithm finds the node, then it will detect that it needs to perform physical removal before returning, as the other search methods do; if it is not found, then another thread has physically unlinked it. Finally, the key and value of the element to move down to the worker-level are returned, which is subsequently handled by the thread in the calling function.

L-DeleteMin. L-DELETEMIN removes the highest priority element from L, which is also the highest priority element in the entire data structure. Like the Lindén-Jonsson algorithm, L-DELETEMIN performs batch physical deletions once some predetermined offset of logically deleted nodes has been reached.

The thread performing L-DELETEMIN first traverses past the prefix of logically deleted nodes in lines 27-35. When the thread successfully marks the first element as logically deleted, which is verified by checking `x_next` in Line 35 (i.e., the return value of the fetch-and-or from Line 34), it exits the loop. If the offset has exceeded the maximum offset, the thread then resets the head's next pointer to the node just marked for logical deletion (lines 38-39). Note that, unlike the Lindén-Jonsson algorithm, we do not need to perform a compare-and-swap to unlink the prefix since only one thread is ever performing a delete-min operation at the time. Nonetheless, it is still beneficial for performance to delay physical deletion, as concurrent insertions face fewer cache misses when traversing the list.

5.2 Worker-Level Min-Heap

The min-heap is implemented as a pre-allocated array. If it becomes full, an additional array is allocated and linked to the first array. Each min-heap is guarded by a single lock. There are only ever up to two threads competing for a heap H_p's lock: the thread p, which is local to the heap, and the thread that is the coordinator performing a delete-min operation. In practice, however, we find it very uncommon for a coordinator thread to need access to the worker level thanks to our helping mechanism, as detailed in Section 3.

Algorithm 1 Pseudocode for the interface methods of the leader-level linked list.

```
1: function L-Insert(list_t list, Node *lead_largest,
   key_t key, val_t val, int tid)
2:   repeat
3:     l_node, r_node ← search(list, key)
4:     new_node ←
           NewNode(key, val, tid, r_node)
5:     if CAS(&l_node.next, r_node, new_node)
         then
6:       if not *lead_largest or
           key > *lead_largest.key then
7:         *lead_largest = new_node
8:       return
9:   until True
10:
11: function L-DeleteMaxP(list_t list,
    Node *lead_largest, int tid)
12:   start_lead_largest ← *lead_largest
13:   repeat
14:     l_ptrs ← (lead_largest, start_lead_largest)
15:     l_node, r_node ←
            searchDelete(list, l_ptrs, tid)
16:     r_node_next ← r_node.next
17:     if not is_marked_ref(r_node_next) then
18:       if CAS(&r_node.next, r_node_next,
            get_moving_ref(r_node_next)) then
19:         break
20:   until True
21:   if not CAS(&l_node.next, r_node,
        r_node_next) then
22:     searchPhysDel(list, r_node)
23:   return [r_node.key, r_node.val]
24: function L-DeleteMin(list_t list)
25:   offset ← 0
26:   x ← list.head
27:   repeat
28:     offset ← offset + 1
29:     x_next ← x.next
30:     if get_notlogdel_ref(x_next) = list.tail
          then
31:       return EMPTY
32:     if is_logdel_ref(x_next) then
33:       continue
34:     x_next ← fetch_and_or(&x.next, 1)
35:   until x ← get_notlogdel_ref(x_next) and
        not is_logdel_ref(x_next)
36:   new_head ← x
37:
38:   if offset > MAX_OFFSET then
39:     list.head.next ←
            get_logdel_ref(new_head)
40:   return [x.key, x.val, x.tid]
```

6 Correctness

A sketch of the proof of PIPQ's linearizability is provided in Appendix B due to space constraints. The complete proof and an analysis of PIPQ's single-threaded worst-case performance are included in the extended version of the paper.

7 Evaluation

We conduct our experiments on a machine running Ubuntu 22.04.3 LTS equipped with four Intel Xeon Platinum 8160 processors containing a total of 96 cores split between four NUMA nodes. Our code is written in C++ and compiled with `-std=c++20 -O3 -mcx16`. Our data points represent the average of three trials performed for five seconds each. Threads are pinned to cores using a NUMA node by NUMA node policy. Hyperthreading is disabled.

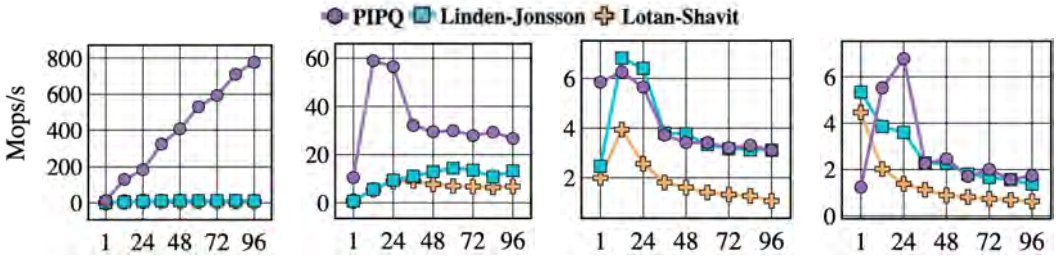
We compare PIPQ with the following two state-of-the-art, strict priority queues: Lotan-Shavit [28] and Lindén-Jonsson [18]. Lotan-Shavit [28] is a priority queue implemented on top of Fraser's skiplist [11]. The delete-min operation is implemented by traversing the lowest-level linked list, and atomically marking the first unmarked node as logically deleted. This data structure is not linearizable but is quiescently consistent [16]. Lindén-Jonsson [18] extends the Lotan-Shavit design by batching the physical deletions once a certain threshold of logically deleted nodes has been met. Lindén-Jonsson outperforms Sundell and Tsigas' linearizable priority queue algorithm [29] and Herlihy and Shavit's [16] lock-free adaptation of the LS priority queue [28].

We test different applications, workloads, and data access patterns: a traditional microbenchmark to evaluate concurrent data structures; a designated thread experiment, where each thread performs either insert *or* delete-min, but not both; and a phased experiment, in which the workload changes in phases. We additionally evaluate the single-source shortest path graph algorithm, which is included only in the extended version of the paper.

7.1 Microbenchmark

Results for the microbenchmark are shown in Figures 3, 4 and 5 where we evaluate throughput (millions of operations per second), latency (microseconds), and which path is followed by PIPQ's insert operations, respectively, as we increase threads. In our microbenchmark, threads perform a series of insert and delete-min operations based on the given mixed workload. The key and value are integers ranging between 1 and 100M and are randomly generated using a uniform distribution. We include four workloads: 100% insert, 95% insert, 50% insert, and 100% delete-min operations. Via experimentation, we find that using values of 10 for `CNTR_MIN` and 100 for `CNTR_MAX` achieves the goal of maintaining a proper size of the leader-level linked list, as discussed in Section 3.

We also performed experiments (not included in the paper) in which we tracked the average number of nodes traversed by the two functions that traverse L past its logically deleted prefix of nodes, namely L-Insert and L-DeleteMaxP, for various values of `CNTR_MIN` and `CNTR_MAX`. The following discussion relates to results captured for a 50% inserts workload. Our results show that modifying the value of `CNTR_MAX` has small effect on the length of traversals, whereas modifying the value of `CNTR_MIN` has a clear correlation. Despite this, however, the overall throughput achieved by the runs hardly varied. Thus, we conclude that the traversal on L does not introduce a performance cost on PIPQ.



(a) 100% insert. **(b)** 95% insert. **(c)** 50% insert. **(d)** 100% delete-min.

Figure 3 Throughput using a microbenchmark. The x-axis represents the number of threads.

Throughput Analysis. Refer to Figure 3 for the results containing throughput achieved by PIPQ and its competitors. Overall, PIPQ performs better or as well as Lindén-Jonsson in all experiments, both of which always perform better than Lotan-Shavit. When insertions dominate the workload, the parallel nature of insertions provided by PIPQ showcases the performance advantages of our approach compared to competitors.

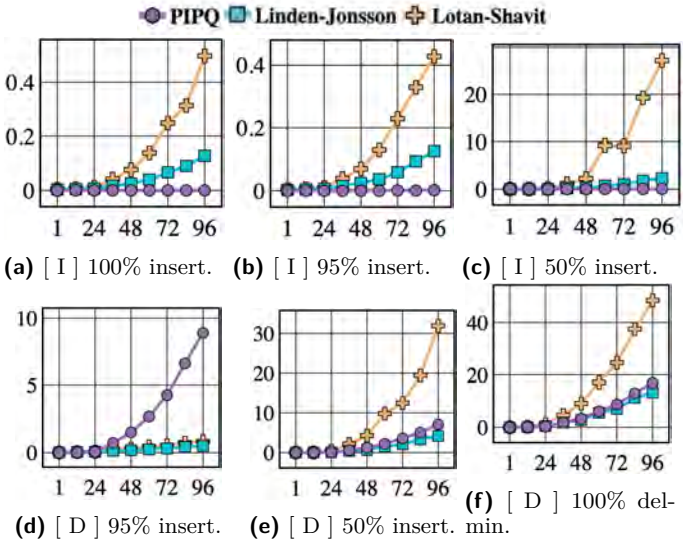
We direct attention to Figure 3a, which shows the 100% insert workload. Since the leader-level linked list remains small, containing up to 100 elements per worker, and there are no threads removing the minimal keys, it becomes highly likely that threads will follow the fast path of the insert operation as more elements are inserted. Thus, insertions become nearly embarrassingly parallel and benefit from cache locality and the speed of the sequential worker min-heap operations, providing significant speedup. At 96 threads, PIPQ performs over 76x better than its closest competitor, Lindén-Jonsson.

Figure 3b shows the results of the 95% insert workload. In this case, we outperform Lindén-Jonsson by almost one order of magnitude at 12 threads (i.e., within a NUMA node) and by 2.4x at 96 threads. To explain the performance drops in the presence of the delete-min operation, we perform an experiment to determine how many operations the coordinator performs on average for the various workloads. We experimentally find that the coordinator

performs, on average, about 24 operations in the case of the 95% insertions workload. Thus, even with a workload of only 5% delete-min operations, nearly every thread in a NUMA node eventually has to wait for the coordinator to complete that thread's operation (or to become the coordinator itself). Because our insert operation is so fast compared to the delete-min operation, potentially many insert requests could be served during the waiting required by the delete-min operation.

That said, however, PIPQ scales significantly better than competitors within a NUMA node, showing the algorithm's ability to increase overall performance for insert-dominant workloads despite such a pessimistic operation (i.e., delete-min) in the mix. The drop in performance after 24 threads is due to the additional synchronization needed to coordinate delete-min operations carried by threads on other NUMA nodes. Nonetheless, PIPQ still outperforms all competitors in the 95% insert case.

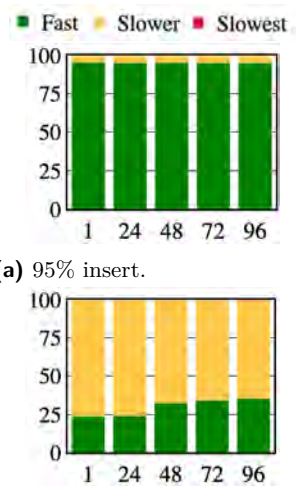
In the 50% insert workload (Figure 3c), PIPQ performs similarly to Lindén-Jonsson, and outperforms Lotan-Shavit. PIPQ also performs similarly to Lindén-Jonsson for the 100% delete-min workload (Figure 3d), though exhibits performance benefits within a NUMA node (i.e., scaling up to 24 threads). Performing so comparably to Lindén-Jonsson for the 100% delete-min workload for 24 threads and beyond shows an interesting correlation. Essentially, the overhead of our multi-level hierarchy, which requires moving elements between levels to ensure strict ordering, is seemingly comparable with the overhead of adjusting Lindén-Jonsson's skip list indexing levels.



(a) [I] 100% insert. **(b)** [I] 95% insert. **(c)** [I] 50% insert.

(d) [D] 95% insert. **(e)** [D] 50% insert. **(f)** [D] 100% del-min.

Figure 4 Latency using microbenchmark (less is better). [I] indicates insert latency, [D] indicates delete-min latency. # of threads in x-axis; microseconds in y-axis.



(a) 95% insert.

(b) 50% insert.

Figure 5 % path is followed by insert operation; # of threads in x-axis.

Latency Analysis. Figure 4 shows the latency of operations for PIPQ and its competitors. The top row of plots shows the average latency of the insert operation for relevant workloads (100% insert, 95% insert, and 50% insert). In each case, PIPQ has the lowest latency compared to competitors by a large margin, revealing the benefit of PIPQ's parallel insertions. For the 100% insertions workload (Figure 4a) PIPQ's insert latency is 265x lower than the latency of Lindén-Jonsson, and in the case of 95% inserts, PIPQ's insert latency is 49x lower than that of Lindén-Jonsson. Finally, in the case of 50% inserts, PIPQ's latency is 40x lower than the latency of Lindén-Jonsson.

The bottom row of the plots in Figure 4 shows the average latency of the delete-min operation. For the 50% insert and 100% delete-min workloads (Figures 4e and 4f, respectively), PIPQ's latency is comparable to that of Lindén-Jonsson, both of which have lower latency than Lotan-Shavit. For the 95% insert workload on the other hand, PIPQ's latency is higher than competitors. Note that PIPQ's delete-min latency at 96 threads in the 95% inserts workload (Figure 4d) is very similar to the latency of delete-min at 96 threads in the 50% workload (Figure 4e), thus revealing that the latency of the operation remains somewhat constant across workloads. This is due to PIPQ's use of combining, and the implicit added cost of Coordinator's operating in a sequential manner.

Path Followed by PIPQ Insert. In order to understand how often the benefit of inserting at the worker-level occurs for various workloads, we perform additional experiments in which we track the path (fast, slower, and slowest) followed by each insert operation. Recall that the fast path is when an insert is to the worker-level. The slow paths involve inserting to the leader-level, and the slowest path requires the additional step of moving an element to the worker-level. Per the results in Figure 5, the slowest path is rarely followed (the color red is hardly visible on the plots); the percentage of the time that it is followed is less than 0.5% for all thread counts in both workloads, significantly less so in many cases.

In the 95% insert workload (Figure 5a), the fast path is followed approximately 95% of the time for all thread counts, and the slower path is followed about 4.9% of the time for all thread counts. In the case of 50% inserts (Figure 5b) the fast path is taken by inserts ranging from 24% of the time with one thread, scaling up to 35% of the time at 96 threads, and the slower path is followed ranging from 76% of the time at one thread, scaling down to 64% of the time at 96 threads. The increased percentage in the slower path being followed for the 50% insert workload compared to the 95% insert workload is simply due to the fact that significantly more elements are being removed in the 50% case, so the probability that an element needs to be inserted to the leader-level given a uniformly generated workload is largely increased compared to that of the 95% case.

7.2 Designated Thread Experiments

Note that designating threads causes calls to Help-Upsert by delete-min operations to be ineffective, since the threads performing delete-min do not ever insert. To address this issue, we move the helping effort to the insert operation, such that after performing its insert, but before it releases its lock on its local H_p, an insert carries out the logic of help-upsert, removing an element from H_p and inserting it into L when necessary. We omit single-threaded performance as it is not a relevant data point for this experiment.

We denote d as the fraction of the total running threads that are designated to perform the delete-min operation. We compared PIPQ with our competitors for values of $d = 1/24$, $1/12$, and $1/2$. Making $d = 1/2$ represents the case where half of all threads perform the delete-min operation, and the other half perform the insert operation. The values of $d = 1/24$ and $1/12$ assign more threads for insertions. The results are in Figure 6. We separated the throughput of insertions and delete-mins into distinct plots to better understand the behavior of each operation.

PIPQ significantly outperforms competitors for insert throughput for all values of d (see Figures 6a, 6b, and 6c). PIPQ achieves the largest speedup for insert throughput in Figure 6a when $d = 1/24$, producing 50x speedup compared to the closest competitor. These experiments show that our insert operation is able to retain performance in the presence

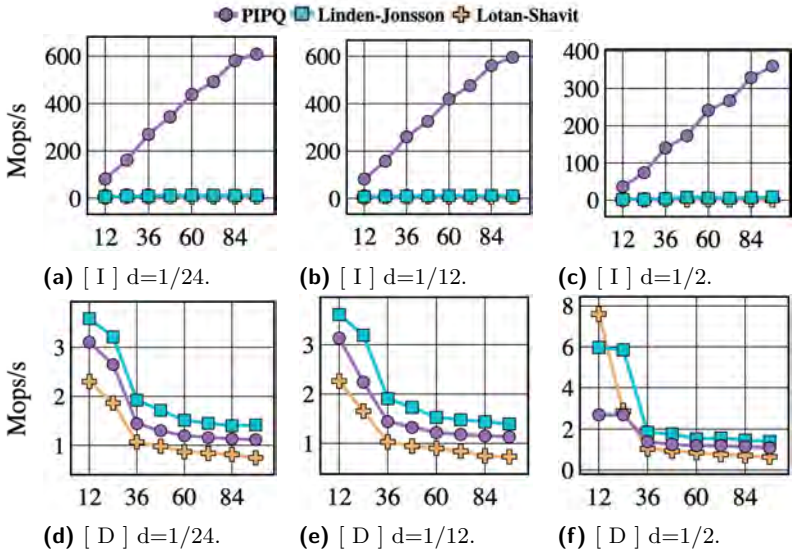


Figure 6 Throughput for the designated thread experiment. "d" denotes the fraction of threads performing delete-min. [I] indicates insert throughput, [D] indicates delete-min throughput. The x-axis represents the number of threads.

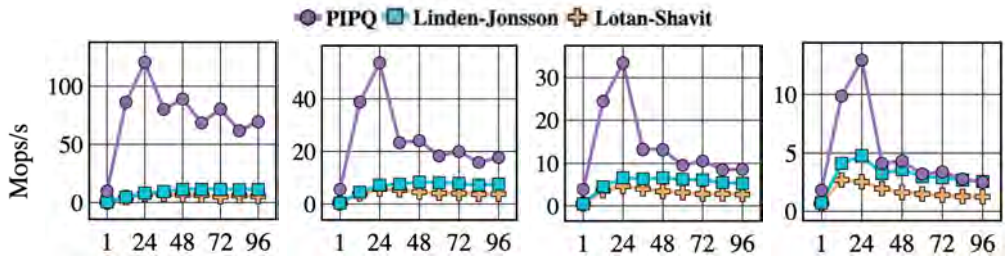
of delete-mins, as long as the threads performing insertions are not stalled due to also performing delete-min operations, as was the case in our microbenchmark experiment. For delete-min, PIPQ achieves higher throughput than Lotan-Shavit and slightly less than that of Lindén-Jonsson. As a result of the high speed of inserts, when we compare the cumulative overall throughput (that is, the summation of the insert and remove-min operations), PIPQ also outperforms its closest competitor by as much as 47x.

Comparing Figures 6f and 6c with Figure 3c exemplifies the effect of dedicating threads for each operation. In Figure 3c, it is expected that half of the threads perform delete-min, and half perform insert at any given time (similar to Figures 6f and 6c). However, PIPQ retains higher performance of insert when threads do not alternate between the two operations.

7.3 Phased Experiment

Some applications of priority queues involve periods of only insertions followed by periods of only delete-mins. We define this workload as "phased" to represent the various periods experienced by the application. During each phase, the total work is split among active threads. The results are shown in Figure 7. The experiments consist of the following phases: in each experiment, 50 million elements are inserted into the priority queue in Phase 1. Once all elements have been inserted, in Phase 2, we remove 1 million (Figure 7a), 5 million (Figure 7b), 10 million (Figure 7c), and 50 million (Figure 7d), i.e. all, elements.

PIPQ outperforms competitors in the three experiments that only remove a fraction of the elements inserted, significantly so within a NUMA node (i.e., up to 24 threads). In each case, Lindén-Jonsson is the closest competitor. As the number of delete-min operations performed in Phase 2 increases, the gap in performance becomes smaller. This aligns with one of our initial motivations that PIPQ's design fits applications where inserts outnumber deletes. In Figure 7a, PIPQ achieves the largest performance gains, as expected, since insertions most significantly dominate the workload in this experiment. Specifically, PIPQ produces up to 15x speedup compared to Lindén-Jonsson at 24 threads and 6.3x speedup compared to



(a) 50->1. **(b)** 50->5. **(c)** 50->10. **(d)** 50->50.

Figure 7 Phased experiments. Workloads are denoted "A->B", meaning Phase 1 is A million inserts and Phase 2 is B million delete-mins. The x-axis represents threads.

Lindén-Jonsson at 96 threads. In Figure 7b, which shows removing 10% of inserted items, 7.4x speedup is produced compared to Lindén-Jonsson at 24 threads, and 2.3x speedup at 96 threads. For the case of removing 20% of the inserted items, shown in Figure 7c, PIPQ gains 5.1x speedup over Lindén-Jonsson at 24 threads and 1.6x speedup at 96 threads.

In Figure 7d, we show when 100% of insertions are then removed. Similar to before, the lesser gap in performance compared to Figures 7a, 7b, and 7c is due to the asymmetry in performance between our two operations; since our insertions are very fast, the amount of time consumed during the second phase is greater compared to the first phase, and so the effect of the delete-min becomes the overwhelmingly dominating factor in performance. That said, however, PIPQ still gains 2.7x speedup over Lindén-Jonsson within a NUMA node.

8 Conclusion

In this paper we presented PIPQ, a strict, linearizable concurrent priority queue. Its design focuses on increasing the performance and parallelism of insert operations as opposed to focusing on the sequential bottleneck of delete-min operations, as done by the majority of prior work. PIPQ achieves this by allowing insert operations to utilize per-thread data structures in the common case. Our evaluation confirms that with PIPQ's design, it is possible to achieve high performance and scalability under various access patterns and runtime configurations.

References

1 Dan Alistarh, Justin Kopinsky, Jerry Li, and Nir Shavit. The spraylist: a scalable relaxed priority queue. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 11–20, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2688500.2688523.

2 Thaïs Baudon, Gabriel Radanne, and Laure Gonnord. Bit-stealing made legal: Compilation for custom memory representations of algebraic data types. *Proc. ACM Program. Lang.*, 7(ICFP):813–846, 2023. doi:10.1145/3607858.

3 Mohamed Benaïchouche, Van-Dat Cung, Salah Dowaji, Bertrand Le Cun, Thierry Mautor, and Catherine Roucairol. Building a parallel branch and bound library. In Afonso Ferreira and Panos M. Pardalos, editors, *Solving Combinatorial Optimization Problems in Parallel - Methods and Techniques*, volume 1054 of *Lecture Notes in Computer Science*, pages 201–231. Springer, 1996. doi:10.1007/BFB0027123.

4 Anastasia Braginsky, Nachshon Cohen, and Erez Petrank. Cbpq: High performance lock-free priority queue. In *Proceedings of the 22nd International Conference on Euro-Par 2016:*

Parallel Processing - Volume 9833, pages 460–474, Berlin, Heidelberg, 2016. Springer-Verlag. doi:10.1007/978-3-319-43659-3_34.

5 Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. Odyssey: A journey in the land of distributed data series similarity search. *Proc. VLDB Endow.*, 16(5):1140–1153, January 2023. doi:10.14778/3579075.3579087.

6 Yanhao Chen, Fei Hua, Yuwei Jin, and Eddy Z. Zhang. Bgpq: A heap-based priority queue design for gpus. In *ICPP 2021: 50th International Conference on Parallel Processing, Lemont, IL, USA, August 9 - 12, 2021*, ICPP '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3472456.3472463.

7 Dave Dice, Virendra J. Marathe, and Nir Shavit. Flat-combining numa locks. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 65–74, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989493.1989502.

8 Panagiota Fatourou, Nikos Giachoudis, and George Mallis. Highly-efficient persistent fifo queues. In *Structural Information and Communication Complexity: 31st International Colloquium, SIROCCO 2024, Vietri Sul Mare, Italy, May 27–29, 2024, Proceedings*, pages 238–261, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-60603-8_14.

9 Panagiota Fatourou and Nikolaos D. Kallimanis. Revisiting the combining synchronization technique. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 257–266, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2145816.2145849.

10 Panagiota Fatourou, Nikolaos D. Kallimanis, and Eleftherios Kosmas. The performance power of software combining in persistence. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '22, pages 337–352, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3503221.3508426.

11 Keir Fraser. *Practical lock-freedom*. PhD thesis, University of Cambridge, UK, 2004. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.599193.

12 Olivia Grimes, Ahmed Hassan, Panagiota Fatourou, and Roberto Palmieri. PIPQ Priority Queue. Software (visited on 2025-10-13). URL: https://github.com/sss-lehigh/pipq, doi:10.4230/artifacts.24906.

13 Binglei Guo, Jiong Yu, Dexian Yang, Hongyong Leng, and Bin Liao. Energy-efficient database systems: A systematic survey. *ACM Comput. Surv.*, 55(6):111:1–111:53, 2023. doi:10.1145/3538225.

14 Timothy L. Harris. A pragmatic implementation of non-blocking linked-lists. In *Proceedings of the 15th International Conference on Distributed Computing*, DISC '01, pages 300–314, Berlin, Heidelberg, 2001. Springer-Verlag. doi:10.1007/3-540-45414-4_21.

15 Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In *Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '10, pages 355–364, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1810479.1810540.

16 Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, April 2008.

17 Yuchen Li, Shixuan Sun, Hanhua Xiao, Chang Ye, Shengliang Lu, and Bingsheng He. A survey on concurrent processing of graph analytical queries: Systems and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024. doi:10.1109/TKDE.2024.3393936.

18 Jonatan Lindén and Bengt Jonsson. A skiplist-based concurrent priority queue with minimal memory contention. In *Principles of Distributed Systems. OPODIS 2013*, pages 206–220. Springer International Publishing, December 2013. doi:10.1007/978-3-319-03850-6_15.

19 Xiangzhen Ouyang and Yian Zhu. Core-aware combining: Accelerating critical section execution on heterogeneous multi-core systems via combining synchronization. *J. Parallel Distrib. Comput.*, 162(C):27–43, April 2022. doi:10.1016/j.jpdc.2022.01.001.

20 Botao Peng, Panagiota Fatourou, and Themis Palpanas. Fast data series indexing for in-memory data. *VLDB J.*, 30(6):1041–1067, 2021. doi:10.1007/S00778-021-00677-2.

21 Botao Peng, Panagiota Fatourou, and Themis Palpanas. Paris+: Data series indexing on multi-core architectures. *IEEE Trans. Knowl. Data Eng.*, 33(5):2151–2164, 2021. doi:10.1109/TKDE.2020.2975180.

22 Botao Peng, Panagiota Fatourou, and Themis Palpanas. SING: sequence indexing using gpus. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1883–1888. IEEE, 2021. doi:10.1109/ICDE51399.2021.00171.

23 Stefano Petrangeli, Jeroen Van Der Hooft, Tim Wauters, and Filip De Turck. Quality of experience-centric management of adaptive video streaming services: Status and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(2s), May 2018. doi:10.1145/3165266.

24 Anastasiia Postnikova, Nikita Koval, Giorgi Nadiradze, and Dan Alistarh. Multi-queues can be state-of-the-art priority schedulers. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '22, pages 353–367, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3503221.3508432.

25 William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, June 1990. doi:10.1145/78973.78977.

26 Hamza Rihani, Peter Sanders, and Roman Dementiev. Multiqueues: Simple relaxed concurrent priority queues. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 80–82, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2755573.2755616.

27 Konstantinos Sagonas and Kjell Winblad. The contention avoiding concurrent priority queue. In Chen Ding, John Criswell, and Peng Wu, editors, *Languages and Compilers for Parallel Computing*, pages 314–330, Cham, 2017. Springer International Publishing.

28 N. Shavit and I. Lotan. Skiplist-based concurrent priority queues. In *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, pages 263–268, 2000. doi:10.1109/IPDPS.2000.845994.

29 H. Sundell and P. Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 11 pp.–, 2003. doi:10.1109/IPDPS.2003.1213189.

30 Martin Wimmer, Jakob Gruber, Jesper Larsson Träff, and Philippas Tsigas. The lock-free k-lsm relaxed priority queue. *SIGPLAN Not.*, 50(8):277–278, January 2015. doi:10.1145/2858788.2688547.

31 Deli Zhang and Damian Dechev. A lock-free priority queue design based on multi-dimensional linked lists. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):613–626, 2016. doi:10.1109/TPDS.2015.2419651.

32 Guozheng Zhang, Gilead Posluns, and Mark C. Jeffrey. Multi bucket queues: Efficient concurrent priority scheduling. In Kunal Agrawal and Erez Petrank, editors, *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2024, Nantes, France, June 17-21, 2024*, pages 113–124. ACM, 2024. doi:10.1145/3626183.3659962.

33 Tingzhe Zhou, Maged Michael, and Michael Spear. A practical, scalable, relaxed priority queue. In *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3337821.3337911.

A Algorithm details of PIPQ

A.1 Structures and Definitions

Algorithms 2 and 3 contain definitions, variables, and structures regarding the worker-level and PIPQ as a whole. Note that not all details are provided, but what is present should be sufficient to properly understand the provided pseudocode.

Algorithm 2 Worker-level min-heap and leader-level linked list definitions.

```
1: struct Heap
2:     list : pointer to HeapNode
3:     size : integer
4:     lock : pointer to integer

5: struct HeapNode
6:     key : integer                                   ▷ the priority of the element
7:     val : val_t

8: struct LeaderListNode
9:     key : integer                                   ▷ the priority of the element
10:    val : val_t
11:    tid : integer                                   ▷ the thread identifier
12:    next : pointer to LeaderListNode

13: struct LeaderList
14:    head : pointer to LeaderListNode (sentinel)
15:    tail : pointer to LeaderListNode (sentinel)
16:    max_offset : int
```

Algorithm 3 PIPQ Implementation.

```
1: struct PIPQ
2: ▷ The leader-level linked list, and array of worker-level min-heaps ◁
3:    worker_heaps[THREADS] : pointer to Heap
4:    leader_list : LeaderList
5:
6: ▷ Locks protecting the Leaders (one per NUMA node) and the Coordinator ◁
7:    compete_coord_locks[NUM_NUMA_NODES] : pointer to integer
8:    coord_lock : pointer to integer
9:
10:   announce[THREADS] : AnnounceStruct          ▷ Announcement for DeleteMin combining
11:   lead_largest_ptrs[THREADS] : pointer to LeaderListNode   ▷ Least priority element in L
12:   leader_counters[THREADS] : pointer to integer            ▷ Number of elements in L
13:
14:   HLS, THREADS, CNTR_MIN, CNTR_MAX, MAX_OFFSET : constant integer
15:
16:   PIPQ(heap_size, nthreads, cntr_min, cntr_max, max_offset) :
17:     HLS(heap_size)
18:     THREADS(nthreads)
19:     CNTR_MIN(cntr_min)
20:     CNTR_MAX(cntr_max)
21:     MAX_OFFSET(max_offset)
```

A.2 Adapted Harris Search Functions for L

Recall that SEARCH (Algorithm 4) is used to support the L-INSERT function of the leader-level linked list, returning the two nodes to insert between (`l_node` and `r_node`). Text marked in red in Algorithm 4 denote modifications made to the original Harris search algorithm.

In lines 8-17, SEARCH must first pass by the prefix of logically deleted nodes, and then when finding `l_node` and `r_node`, account for the fact that a concurrent operation may have marked a node between the left and right nodes as moving, but not yet performed physical deletion (see the check in Line 21). If this is the case, a compare-and-swap operation in Line 27 of SEARCH is performed to carry out the physical deletion. Finally, `l_node` and `r_node` are returned to L-INSERT.

Now direct attention to Algorithm 5, SEARCHDELETE. Recall that SEARCHDELETE supports the L-DELETEMAXP function of the leader-level linked list, which removes the least priority element tagged with some given $\texttt{tid}_\texttt{p}$, to be moved down to the worker level. The function begins by traversing L until it encounters `start_lead_largest` (i.e., the last node in L tagged with the relevant thread identifier). Note that in L-DELETEMAXP we must store pointer `start_lead_largest` (which initially points to the same node as `lead_largest`), in case `lead_largest` is reset in Line 21, and then the subsequent CAS in Line 28 fails and the function must be re-run.

Algorithm 4 Adapted Harris search, used to support the L-INSERT method (see Algorithm 1) of the leader-level linked list.

```
1: function SEARCH(list_t list, int key)
2:     repeat
3:         ▷ 1. Find l_node and r_node ◁
4:         l_node, r_node
5:         prev_log_del ← False
6:         x ← list.head
7:         x_next ← x.next
8:         repeat
9:             if not IS_MOVING_REF(x_next) then
10:                l_node ← x
11:                l_node_next ← GET_NOTLOGDEL_REF(x_next)
12:            x ← GET_UNMARKED_REF(x_next)
13:            if x = list.tail then
14:                break
15:            prev_log_del = IS_LOGDEL_REF(x_next)
16:            x_next ← x.next
17:        until x.key ≥ key and not IS_MOVING_REF(x_next) and not prev_log_del
18:        r_node ← x
19:
20:        ▷ 2. Check if l_node and r_node are adjacent ◁
21:        if l_node_next = r_node then
22:            if IS_LOGDEL_REF(l_node.next) or (r_node.next and IS_MOVING_REF(r_node.next)) then
23:                continue
24:            return [l_node, r_node]
25:
26:        ▷ 3. Remove one or more "moving" nodes ◁
27:        if CAS(&l_node.next, l_node_next, r_node) then
28:            if IS_LOGDEL_REF(l_node.next) or (r_node.next and IS_MOVING_REF(r_node.next)) then
29:                continue
30:            return [l_node, r_node]
31:    until True
```

During traversal, SEARCHDELETE tracks the nearest predecessor of the node pointed to by `start_lead_largest` that is also tagged with $\texttt{tid}_p$ in variable `new_lead_largest`, in order to reset `lead_largest`. Once the traversal is complete (i.e., `start_lead_largest` has been found), `lead_largest` is reset to `new_lead_largest` (Line 21 of SEARCHDELETE). Note that access to `lead_largest` is protected by the underlying lock acquired on the associated thread's heap. As in SEARCH, lines 24-29 of the algorithm ensures that there are not any marked nodes between `l_node` and `r_node` (removing any if found), before finally returning the two pointers.

The final search algorithm is SEARCHPHYSDEL. The algorithm differs only slightly from SEARCH (Algorithm 4), such that it searches for a specific node instead of the location to insert a specific key. Its only goal is to ensure physical deletion of `search_node`, and thus does not return node pointers. The pseudocode is included in the extended version of the paper.

Also included in the extended version of the paper is pseudocode for the following algorithms: WORKER-INSERT, WORKER-DELETE-MIN, HELP-UPSERT, and the insert and delete-min APIs of PIPQ.

B Correctness

We assume a typical asynchronous system with n threads which communicate by accessing shared variables. To prove the correctness of PIPQ, we break it down to its two-level components: the worker level and the leader level. At the worker level, each thread p has its own min-heap H_p protected by a single global lock l_p. Lock l_p also protects access to $lead_largest_p$. At the leader level, PIPQ maintains a linked list L of keys. Consider an execution α of PIPQ. All our claims below are in reference to this execution.

Algorithm 5 Adapted Harris search, used to support the L-DeleteMaxP method (see Algorithm 1) of the leader-level linked list.

```
1: function searchDelete(list_t list, node_t start_node,
   pair_t lead_largests = (lead_largest, start_lead_largest), int tid)
2:     x, l_node, r_node ← start_node
3:     x_next ← x.next
4:     repeat
5:         ▷ 1. Find l_node and r_node ◁
6:         while True do
7:             if not is_moving_ref(x_next) then
8:                 cur_l_node ← x
9:                 cur_l_node_next ←
                       get_notlogdel_ref(x_next)
10:            x ← get_unmarked_ref(x_next)
11:            if x = list.tail then
12:                break
13:            x_next ← x.next
14:            if not is_moving_ref(x_next) and
                   x.tid = tid then
15:                new_lead_largest ← r_node
16:                l_node ← cur_l_node
17:                l_node_next ← cur_l_node_next
18:                r_node ← x
19:                if x = start_lead_largest then
20:                    break
21:        *lead_largest ← new_lead_largest
22:
23:        ▷ 2. Check if l_node and r_node are adjacent ◁
24:        if l_node_next = r_node then
25:            return [l_node, r_node]
26:
27:        ▷ 3. Remove one or more "moving" nodes ◁
28:        if CAS(&l_node.next, l_node_next, r_node) then
29:            return [l_node, r_node]
30:        x ← list.head
31:        x_next ← x.next
32:        r_node ← new_lead_largest
33:    until 1
```

Recall that L contains two sentinel nodes, one storing the key $-\infty$ pointed to by pointer *head*, and another storing the key $+\infty$ pointed to by the pointer *tail*. The *elements* (or *nodes*) of L at some point t are the keys (nodes) that are traversed if we start from the node pointed to by *head* at t and we move forward following the value (at t) of the next pointer of each node. We call p-element an element of L which is tagged with p (i.e., the *tid* field of the element's struct has the value p). Initially, L contains just the two sentinel nodes and it is sorted.

The complete proof of PIPQ's linearizability is included in the extended version of the paper due to space limitations. To show the conceptual flow of the proof, in the following we list the definitions, invariants, lemmas, and corollaries formally discussed in the extended version of the paper.

▶ **Lemma 1.** *For each thread p,*

1. *H_p is a linearizable priority queue; each worker operation that is applied on H_p is atomic (as it is applied while the thread that executes it holds the lock l_p);*
2. *All accesses to $lead_largest_p$ are performed atomically (while lock l_p is held). Whenever $lead_largest_p$ is read, it points to the p-element with the largest key in L among all p-elements.*
3. *While p executes an* L-Insert *to insert a p-element in L, it holds the lock l_p; moreover, the inserted element is of a greater or equal priority than that of the highest priority element in p's worker-level heap, H_p, at the time that p acquired l_p;*
4. *While p inserts a p-element into L it holds the lock l_p;*
5. *Every key that appears as an argument of some* PIPQ-Insert *operation invoked by p (in α) is inserted either in H_p or in L (and not in both).*

▶ **Lemma 2.** *For each thread p, whenever a* L-DeleteMin *is invoked, if H_p is not empty, then the value of L_count_p is greater than or equal to 2 indicating that there are at least two active p-elements in L.*

▶ **Lemma 3 ([14], Sections 4.1 and 5.1).** *Consider any instance I of* Search *with key k. Let l_node and r_node be the pointers returned by I; call* left node *and* right node *the nodes pointed to by these pointers, respectively. Then, there is some point t during the invocation and the response of I, at which all the following conditions hold:*

1. *the left node and the right node are nodes of L at t,*
2. *the key of the left node must be less than k (unless k is the highest priority in L, in which case the left node may be a logically deleted node whose key is greater than or equal to k) and the key of the right node must be greater than or equal to k,*
3. *neither node is marked as moved and the right node is not logically deleted, and*
4. *the right node must be the immediate successor of the left node in L.*

▶ **Invariant 4.** *L is a list whose last element is the sentinel node pointed to by tail.*

▶ **Lemma 5.** *An instance of* L-DeleteMin *and an instance of* L-DeleteMaxP *that are concurrent never attempt to delete the same node in L.*

▶ **Corollary 6.** *A node in L will never have both its DELMIN and its MOVING bit marked.*

▶ **Invariant 7.** *The following claims hold:*

1. *The logically deleted nodes in L form a prefix of the list;*
2. *The suffix of L that is comprised of nodes that are not logically deleted is sorted;*

▶ **Corollary 8.** *The element with the highest priority in L is its leftomost (first) active node.*

▶ **Lemma 9.** *The linearization point of each* L-Insert, L-DeleteMin, *or* L-DeleteMaxP *is within the execution interval of the operation.*

▶ **Lemma 10.** *Denote by ℓ_L the linearization order we derive by the linearization points of the* L-Insert, L-DeleteMin, *and* L-DeleteMaxP *operations (of α). The response of a* L-DeleteMin *or a* L-DeleteMaxP *operation is consistent with respect to ℓ_L.*

▶ **Corollary 11.** *The implementation of L is linearizable.*

▶ **Lemma 12.** *The linearization point of each* PIPQ-Insert *and* PIPQ-DeleteMin *operation op of PIPQ is within the execution interval of op.*

▶ **Definition 13.** *Denote by ℓ_H the linearization of the worker operations (of α). Define $\ell_H(p,t)$ (for a specific thread p), as the subsequence of ℓ_H containing those worker operations that have been linearized by a fixed time t. For each p, the abstract state of H_p at t is the set of elements contained in the priority queue that results when the operations in $\ell_H(p,t)$ are applied* sequentially *on an initially empty priority queue.*

▶ **Definition 14.** *Denote by $\ell_L(t)$ the prefix of ℓ_L containing those operations on L that have been linearized by a fixed time t. The abstract state of L at t is the set of elements contained in the singly-linked list that results when the operations in $\ell_L(t)$ are applied* sequentially *on an initially empty linked list.*

▶ **Invariant 15.** *For each process p, the keys of the p-elements contained in the abstract state of L are the smallest among all the p-elements in the union of the sets that comprise the abstract state of H_p and the abstract state of L.*

▶ **Definition 16.** *At each point t, the abstract state of PIPQ is defined by the (multi-set) union of the abstract states of H_1, H_2, ..., H_n and the abstract state of L.*

▶ **Lemma 17.** *Denote by ℓ the linearization order we derive by the linearization points of the* PIPQ-Insert *and* PIPQ-DeleteMin *operations (of α). The responses of* PIPQ-DeleteMin *operations in PIPQ are consistent with respect to ℓ.*

▶ **Theorem 18.** *PIPQ is a linearizable implementation of a priority queue.*

C Complexity Analysis and Discussion

Given that PIPQ is blocking, and the leader-level linked list is lock-free (not wait-free), it is impossible to give a worst-case bound for either. However, for completeness, it is worth discussing the single-threaded worst-case bound, which is $O(lg(m) + k)$ for both insert and delete-min, where m is the size of the relevant worker-level heap, and k is the size of the leader-level linked list. This is directly inherited from the worst-case performance of its components.

We believe the above worst-case analysis is not a practical issue, due to several reasons. First, for delete-min, this worst case will only occur when the coordinator must upsert an element from a worker heap (i.e., when it removes an element and finds that its relevant counter value is less than 2). In all other cases, the coordinator removes the highest-priority element, which is the first non-logically-deleted element in the list. Also, our helping mechanism offloads most of this "upsert" mechanism to the pending delete-min operations that are waiting for the coordinator, which further reduces the probability that the coordinator will ever need to "upsert" elements. For insert, the worst case represents the scenario in which an element is inserted at the leader level, and an element is subsequently moved down to the worker level. Similar to delete-min, as shown in Figure 5, this scenario (which we call the "slowest path") rarely occurs in practice.

Even when this worst case scenario is unavoidable, our design mitigates its effects by configuring `CNTR_MIN` and `CNTR_MAX`. In particular, reducing `CNTR_MAX` makes the size of the linked list as small as the number of threads (which is practically constant and comparable to the cost of any alternative with logarithmic-time performance). Also, by increasing `CNTR_MIN` we increase the chance that helper threads upsert elements ahead of the coordinator, which reduces the likelihood of this worst-case scenario. In fact, some of our experimental trials that we did not include in the paper aimed at assessing the effect of replacing the leader-level linked list with other data structures, and we observed no performance gains in any of them.

LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics

Martin Hilgendorf
Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

Marina Papatriantafilou
Chalmers University of Technology, Gothenburg, Sweden
University of Gothenburg, Sweden

Abstract

Data sketches balance resource efficiency with controllable approximations for extracting features in high-volume, high-rate data. Two important points of interest are highlighted separately in recent works; namely, to (1) answer multiple types of queries from a single data structure built in one pass over the data, and (2) perform both queries and updates concurrently. In this work, we now tackle the new challenges arising when combining these useful directions together.

We investigate the trade-offs around efficiency, consistency, and accuracy to be balanced and synthesize key ideas into LMQ-Sketch, a single, composite data sketch supporting concurrent updates and multiple queries (frequency point queries, frequency moments F_1, and F_2 as representative selection). Our method "Lagom" is a cornerstone of LMQ-Sketch for low-latency global querying (<100 µs), combining freshness, timeliness, and accuracy with a low memory footprint and high throughput (>2B updates/s). We analyze and evaluate the accuracy of Lagom, which builds on a simple geometric argument and efficiently combines work distribution with synchronization for proper concurrency semantics – *monotonicity of operations* and *intermediate value linearizability*. Comparing with state-of-the-art methods, which, as mentioned, provide either mixed queries or concurrency separately, LMQ-Sketch shows highly competitive throughput, with additional accuracy guarantees and concurrency semantics, while also reducing the required memory budget by an order of magnitude. We expect the methodology to have broader impact on concurrent multi-query sketches.

2012 ACM Subject Classification Computing methodologies → Shared memory algorithms; Theory of computation → Concurrent algorithms; Information systems → Data structures

Keywords and phrases Concurrent Data Structures, Data Sketches, IVL, Freshness, Synchronization

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.36

Related Version *Previous Version*: `https://arxiv.org/abs/2506.16928`

Supplementary Material *Software*: `https://gitlab.com/marhilg/lmq-sketch` [26]
archived at `swh:1:dir:d33d3a06fac57f29b697bc57c19fa414b6f7c8cb`

Funding Work supported by the Swedish Research Council project EPITOME 2021-05424, by the Marie Skłodowska-Curie Doctoral Network RELAX-DN, funded by the European Union under Horizon Europe 2021-2027 Framework Programme (grant nr. 101072456) and by Chalmers University AoA frameworks Energy and Production, WP. INDEED, and "Scalability, Big Data and AI".

1 Introduction

Data sketches provide approximate summaries of data streams and can answer questions of interest efficiently, with bounded memory requirements. Examples include estimation of element frequencies, set/multi-set size, frequency moments (norms), frequent elements, distributions, quantiles, and more [1, 5, 8, 10, 13, 15]. Summarizations involve suitable

© Martin Hilgendorf and Marina Papatriantafilou;
licensed under Creative Commons License CC-BY 4.0

hash functions, and the results are controllable approximations of the targeted aggregate – commonly for skewed data – in form of (ϵ, δ) guarantees (the estimate deviates at most ϵ, with probability at least $1-\delta$). The approximation reflects a *trade-off between accuracy and sketch size*, and, in consequence, the operations' time cost. Due to their usefulness in data analytics and feature extraction, sketches get a major role in data processing platforms [3, 2, 23, 28, 36].

Mixed query sketches. Data analysis requires more information than a single metric. Typically, this would require one sketch for each metric to be tracked [9], requiring additional memory and processing – a prohibitive overhead stressed as early as in [13] – and their contents may be inconsistent. Can it be done differently? Important questions regarding having a *single sketch* for answering *mixed queries* have been asked in works such as [16, 32, 38], showing potential to improve approximation guarantees and quality of results compared to using separate sketches. A notable sketch supporting, with auxiliary data structures, multiple queries is the Count-Min Sketch (CMS) [13, 15]. Besides, at the core of many statistics are *frequency moments* (the i-th such defined as the sum of the i-th power of the frequency of each element). E.g., F_2, also called "surprise number", has uses in linear algebra [45], calculating the Gini index [7, 22], evaluating skewness of distributions, quantiles [20], range queries, heavy-hitters, histograms, or wavelet synopses helping for compression [11, 12, 19, 21]. In conjunction with other metrics, F_2 enables to detect anomalies [10, 29, 33, 43] and associates with universal sketching [4, 31]. F_2 can be estimated via, e.g., CMS [14] or Fast-AGMS [10].

Concurrency. Given the rates of real-world, continuous streams, multi-threaded processing of updates and concurrent querying is both a *necessity* and a *challenge*, since, on one hand, the perceived order of updates by queries influences accuracy, and, on the other hand, synchronization overhead can dramatically influence throughput, operation timeliness and result freshness. Work to address concurrency among queries and updates on sketching was initiated in recent years, for single-query sketches [18, 27, 34, 42].

Our Targets. We integrate these directions, opening up the arising challenges not tackled by existing approaches in balancing concurrency and accuracy with resource footprint and timeliness, through workload distribution, data structure design, and synchronization for proper consistency guarantees. We target a single, low memory-footprint composite data structure with concurrent updates and queries for a representative selection of queries: F_1, F_2, and point queries (individual element frequencies), with extensibility to other queries in mind.

Challenges. Concurrency adds new factors and trade-offs; the influence is complex: stronger consistency can imply better compliance with sequential sketch accuracy bounds but also higher synchronization overhead, increasing query latency and staleness, in turn aggravating accuracy and throughput. Further, mixed queries should not diverge excessively in observed input to ensure consistency. In short: various trade-offs impact the metrics of interest: set of queries, result quality, resource footprint, operation timeliness, and scalability.

Idea and Contributions. We study these trade-offs in detail and construct *LMQ-Sketch* (*Lagom*[1] *Multi-Query Sketch*), a low memory-footprint sketch supporting concurrent and mixed frequency queries. The input domain and data structure are partitioned to support multiple threads processing updates, a design also present in [42] albeit for point queries only, showing benefits in accuracy and memory-efficiency. However, queries spanning multiple

partitions, *global queries*, face higher operation complexity, synchronization overhead and latency. To this end, our algorithmic design Lagom employs *Intermediate Value Linearizability* (IVL) [39], a modular consistency property suitable for sketches, admitting more efficient implementations compared to requiring linearizability (unnecessarily strong for sketches which approximate by definition) and which can preserve (ϵ, δ) bounds of the sequential counterparts. Lagom demonstrates significant gains in efficiency, timeliness, and result accuracy. It gathers "just enough" information (shown using a geometric argument) from partitions while maintaining accuracy in line with sequential methods. We also show *monotonicity of scans* [17] for cross-query consistency of LMQ-Sketch.

The analysis is complemented by a detailed empirical study using real-world CAIDA data [6] and synthetic stress-tests for extended insights. We compare to state-of-the-art methods: SW-SKT [9] employs separate sketches for multiple, albeit non-concurrent, queries; Delegation Sketch [42] supports concurrent updates and queries (for point queries only). Further, we define elementary baselines based on the competing factors in the trade-offs. Our evaluation shows how LMQ-Sketch efficiently balances memory, accuracy, and concurrency, scaling beyond 2B updates/sec concurrently with high-rate queries. We also propose a methodology to experimentally study the impact of concurrency on accuracy. F_2 query latency between 1–100 µs implies freshness with error below 0.01 %, with a sketch just 4 MiB in size.

Roadmap. After preliminaries (§ 2) and problem analysis (§ 3), we present and analyze the methods constituting LMQ-Sketch (§ 4, 5), detail our empirical evaluation (§ 6, with open-source implementation [26]) and other related work (§ 7), concluding in § 8. Throughout, claims are backed by the main ideas; proof sketches are provided in Appendix A.

2 Background

Count-Min Sketch (CMS) and Frequency Moments. CMS [13] estimates the frequency of keys in the input stream, via *point queries*, with potential overestimation by a factor ϵ with probability $1-\delta$. It uses a matrix of counters, $\mathsf{CMS}[H \times K]$, alongside H hash functions, each mapped to a row. H and K balance memory against accuracy, by $H = \lceil e/\epsilon \rceil$ and $K = \lceil \ln(1/\delta) \rceil$. A CMS update for a key a increments $\mathsf{CMS}[j, h_j(a)]$ for each row j. To answer a point query for a, its frequency $f(a)$ is estimated as $\hat{f}(a) = \min_j \mathsf{CMS}[j, h_j(a)]$.

For a stream S of keys from a domain U, the n-th frequency moment is [46]: $F_n = \sum_{a \in U} f(a)^n$. CMS provides an exact F_1 as $\sum_k^K \mathsf{CMS}[j,k]$ for any row j. To estimate F_2, which gives an indication of the skewness of the frequency distribution, [14] introduces CM^- and CM^+ for different levels of skewness, encoded by the z parameter of a Zipfian, as many real-world phenomena follow Zipf's Law; CM^- for $z \leq 1$, while CM^+ yields better accuracy for skewed data ($z > 1$). Similar to CMS point queries, CM^+ can only overestimate.

▶ **Definition 1** (CM^+ [14]). CM^+ *estimates* F_2 *from a* $\mathsf{CMS}[H \times K]$ *by* $\min_j \sum_k^K \mathsf{CMS}[j,k]^2$ *with relative error* $1+\epsilon$ *with probability* $1 - \delta = 1 - \frac{3}{4}^{-H}$, *where* $\epsilon = O\left(K^{\frac{-(1+z)}{2}}\right)$.

Augmented Sketch (ASketch). ASketch [41] is a CMS extension for skewed streams, where a small subset of keys account for the majority of updates. It uses a *filter*, denoted AF, behaving as a fixed-size map of keys and counters, to count occurrences of the heaviest

[1] Lagom (link) describes "just the right amount", indicating appropriateness rather than suggesting lack.

keys separately from the CMS matrix, improving throughput and accuracy. Alg. 1 shows its operations; `updateAndPQ` (line 1.10) is a simple extension of the CMS update to also get a point query estimate for the key by returning the min of the updated counters. AF can be efficiently searched for a key (line 1.2), e.g., using SIMD; if the key is found, the update increments just a single counter in AF (line 1.3), bypassing calculation of H hash functions and counter increments, and avoiding collisions with light keys in CMS. Keys are swapped between AF and CMS as needed to keep the most frequent keys in the filter (line 1.11). Note that Δ (line 1.13) is the *exact number of occurrences of key a while resident in the filter*.

Algorithm 1 ASketch – with highlighted enhancements for LMQ-Sketch, detailed in § 4.

```
1.1  Function ASketchUpdateEnhanced(key a, value v)
1.2      if a in AF then                                          ▷ Increment count in filter
1.3          AF[a] += v;
1.4          AF^mavg[a] ← wv + (1−w)AF^mavg[a];                   ▷ w = 0.8, bias towards recent changes in skew
1.5      else if AF not full then                                 ▷ Add key to filter
1.6          AF[a] ← v;
1.7          AF^old[a] ← 0;
1.8          AF^mavg[a] ← v;
1.9      else                                                     ▷ Update sketch
1.10         estimatedFreq ← CMS.updateAndPQEnhanced(a, v);
1.11         if estimatedFreq > min_{key in AF}(AF[key]) then
1.12             minKey ← argmin_{key in AF}(AF[key]);
1.13             Δ ← AF[minKey] − AF^old[minKey];
1.14             CMS.updateEnhanced(minKey, Δ);
1.15             AF[a] ← estimatedFreq;
1.16             AF^old[a] ← estimatedFreq;
1.17             AF^mavg[a] ← v;
1.18 Function ASketchQuery(key a)
1.19     if a in AF then return AF[a];
1.20     else return CMS.pointQuery(a);
```

3 Problem Description and Analysis

We target to estimate item frequencies and frequency moments using a single sketch, built from one pass over a high-rate input data stream. The stream consists of tuples (a, v), representing v occurrences of key a; we consider the *cash register* model, so $v > 0$. We target multi-core shared memory systems supporting atomic primitives including fetch-and-add and compare-and-swap. Threads communicate via a coherent memory model, and do not fail or crash. Operations on the sketch object are *updates*, processing input tuples, and *queries* of various kinds, and may all be invoked concurrently by different threads. A solution needs to balance a delicate multi-way trade-off that we analyze in the following.

Accuracy and Consistency. Strong consistency for queries requires atomic views of the sketch state, which may involve significant overhead; weakening these semantics *arbitrarily* may reduce accuracy in unclear ways. We aim for a balance: fresh query results with low synchronization overhead, yet with clear semantics, avoiding the accuracy pitfalls of both. To facilitate this, we build on the idea of *Intermediate Value Linearizability* (IVL) [39], that extends *weak regularity* [35], requiring that a concurrent query return a value in the interval between the minimum and maximum ones it could return in any linearization of the execution. In the cash register model, the considered query values increase monotonically with updates; hence:

▶ **Observation 2.** *For a monotonically increasing function, the minimum return value of query Q, concurrent with updates, is given by an idealized return value of the query (e.g. by a perfect external observer) that observes updates up to the ones completed before Q's start (i.e. no overlapping ones). Similarly, the maximum return value of Q is given by an idealized return value that observes also all updates overlapping Q.*

Q observes all updates completed before it started *exactly once*; the range of permitted return values, effectively the "inaccuracy" accepted as a consequence of IVL, is due to updates concurrent with Q. As we target high data rates, a *short query latency* is critical for minimizing this range and preserving *freshness*. To this end, performance can be improved by further relaxing consistency where tolerable: [39] combines r-relaxation [24] with IVL – allowing Q to also miss up to r updates preceding it – paraphrased here:

▶ **Definition 3.** *A query Q is r-relaxed IVL if it returns a value between the min and max values it could return in any linearization that may reorder Q with up to r operations.*

Multiple Query Types. The targeted queries include *local* and *global* ones. The former involve part of the data, e.g., a point query for a certain key; the latter consider complete contents of the sketch and associate with semantics for bulk operations [17, 35, 37]. Supporting multiple queries also concerns their relative consistency; we identify *operation monotonicity* as a useful notion to tell us how aligned the views of different queries are; paraphrasing from [17, 35]:

▶ **Definition 4** (Monotonicity of scans)**.** *For queries Q_1 and Q_2, where Q_1 precedes Q_2 (denoted $\longrightarrow$), all updates observed (i.e., accounted for) by Q_1 must also be observed by Q_2.*

Accuracy and Memory. The (ϵ, δ) bounds of sketches allow trading more memory for improved accuracy; e.g., in CMS, a query can overestimate due to hash collisions – more counters (or filters as in ASketch) reduce collisions. In the sequential setting, bounds are well-studied, and ideally should be preserved under concurrency. Similarly important are memory *layout and management*. Partitioning the input domain can aid parallelism as partitions can safely be summarized concurrently, enhancing accuracy and efficiency, shown for local queries in [42]. However, aforementioned accuracy-consistency challenges arise for global queries.

Summary: Goals and Challenges. **(G1)** *IVL and associated relaxations* are identified as consistency targets, due to explainability and efficiency advantages for synchronization. *Low query latency* becomes a significant goal, allying with *freshness*. **(G2)** *Monotonicity of scans* is desirable for multiple query types. **(G3)** *Maintaining a single data structure* for answering multiple queries can enable better use of the memory budget if the accuracy for each query can benefit from the full memory. We identify *partitioning* as an ancillary approach to improving accuracy and memory utilization, also aiding parallelization and hence throughput.

4 LMQ-Sketch – Design and Coordination

We describe the design space and coordination in LMQ-Sketch to enable concurrent estimation of multiple queries from a single, composite data structure (Fig. 1) in conjunction with partitioning, as motivated in § 3, for consistency, memory efficiency, and accuracy-friendliness.

The input domain is split into P partitions, as is the data structure (horizontal slices in Fig. 1). Updates are performed by P threads, each "owning" one partition (ASketch and other components) with exclusive update access to it. When thread T_i processes an update for a key owned by another partition, the update is buffered in a corresponding *delegation filter* DF (Alg. 2). Similar to ASketch filters, delegation filters behave as maps of C keys, and are small enough to be searched efficiently with SIMD operations. Periodically (upon a triggering condition, line 2.13) filters are handed over to the owning thread (using, e.g., a Treiber stack [44]), and contents are transferred in bulk to its local ASketch (Alg. 3).

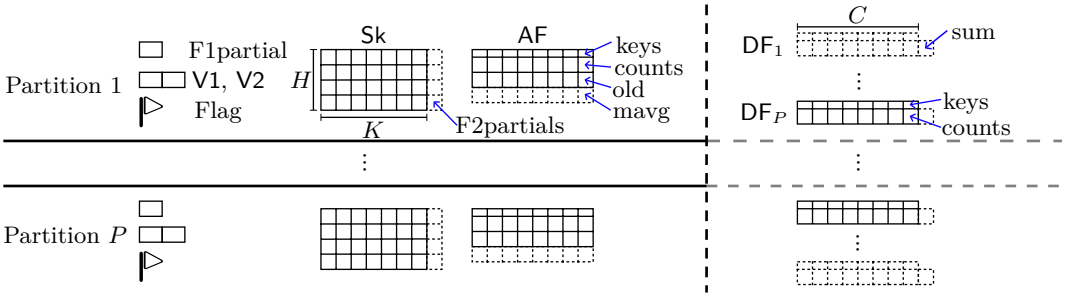


Figure 1 LMQ-Sketch's composite data structure. Partitions, horizontal slices of memory, are modifiable only by the owner thread. All updates eventually reach partition-local ASketches (Sk and AF, managing owned heavy keys), to the left of the vertical boundary. Delegation filters, DF, on the right, buffer updates owned by other partitions and are periodically handed over to the owning thread. C is a parameter, e.g. 16. Dotted components are for optimizations.

Table 1 Notation used throughout.

Symbol	Description
P	No. of partitions & threads
K, H	CMS rows and columns
C	No. of slots in ASketch and delegation filters
B	Max. no. of updates buffered in delegation filter
$\mathsf{Owner}(a)$	Get partition owning key a
T_i	Updater thread for partition i
$T_i.\mathsf{Sk}$	Local sketch for T_i
$T_i.\mathsf{F1partial}$	No. of updates completed by T_i

Symbol	Description
$\mathsf{Sk.F2partials}$	Sum for CM^+ per row (Def. 1)
$T_i.\mathsf{AF}$	ASketch filter for T_i
$\mathsf{AF}[a]$, $\mathsf{AF}^{\mathrm{old}}[a]$	Count & Old count of heavy key a in AF
$\mathsf{AF}^{\mathrm{mavg}}[a]$	Projected count of buffered occurrences of heavy key a
$T_i.\mathsf{DF}_j$	Delegation filter of T_i for updates owned by partition j
$\mathsf{DF}_j[a]$	Count of key a buffered in DF_j
$\mathsf{DF}_j.\mathrm{size}$, $\mathsf{DF}_j.\mathrm{sum}$	Number of keys and updates buffered in DF_j

Algorithm 2 Delegation Sketch update on T_i. Enhancements for LAGOM in green.

```
2.1  Function update(key a, value v)
2.2      filter ← T_i.DF_Owner(a);
2.3      while filter.size = C or filter.sum ≥ B do
2.4          processDelegatedUpdates();
2.5      if a in filter then
2.6          filter[a] += v;
2.7          filter.sum += v;
2.8      else
2.9          filter[a] ← v;
2.10         filter.size += 1;
2.11         filter.sum += v;
2.12     T_i.F1partial += v;
2.13     if filter.size = C or filter.sum ≥ B then
2.14         T_Owner(a).pendingFilters.push(filter);
```

Algorithm 3 Processing delegated updates on T_i. Enhancements for LAGOM in green.

```
3.1  Function processDelegatedUpdates
3.2      while T_i.pendingFilters is not empty do
3.3          Wait until T_i.beingScanned = false;
3.4          T_i.V1++;
3.5          filter ← T_i.pendingFilters.pop();
3.6          foreach a in filter do
3.7              T_i.ASketchUpdateEnhanced(a, filter[a])
3.8          Clear filter contents;
3.9          filter.size ← 0;
3.10         filter.sum ← 0;
3.11         T_i.V2++;
```

Point Queries. The idea of domain-partitioning and delegation is also the basis of the Delegation Sketch [42] for point queries, summarized here for self-containment. Moreover, we here show their consistency properties as a query type in LMQ-Sketch. A point query (Alg. 4) for key a is answered by $T_{\mathsf{Owner}(a)}$, the thread owning a, estimating $\hat{f}(a)$ as the sum of occurrences of a in the thread-local ASketch of $T_{\mathsf{Owner}(a)}$ (line 4.2) and in relevant delegation filters of other threads (line 4.3). Skew in the input data is beneficial to $\hat{f}(a)$'s

accuracy; frequent keys are often present in the filters where they are counted accurately, akin to ASketch. Based on the IVL definition and Obs. 2, along with the fact that entries in the partition-local sketch are increasing with subsequent updates, and that only the thread owning a key performs updates and queries on this sketch (hence cannot miss completed updates or double-count any), we have:

▶ **Lemma 5.** *Delegation Sketch-based PQ is an IVL implementation of ASketch point query.*

4.1 Global Query Trade-offs

This approach of assigning query work to the owner thread will however not work for global queries which span all partitions. We investigate the trade-offs relating concurrency and accuracy for such bulk queries, as identified in § 3: the extremes of the spectrum serve as baselines, followed by our balancing approach.

STRICT: A baseline for strong consistency, using a Readers-Writer lock to allow shared access for updates and point queries, but exclusive access for global queries (acting as writers) to see a *consistent global state*. F_1 is estimated by summing all counters on any row of each T_i.Sk and counts in all ASketch and delegation filters, yielding the number of updates completed before the query. F_2 can be estimated using CM^+ by merging all sketches and flushing filter contents (treating them as updates), thus preserving its error bounds. Note several drawbacks: updates incur locking overhead even when no global query is taking place; further, while thread-safe, the result can be stale due to "stopping the world" – input tuples keep arriving but are not processed or visible to the query.

NOSYNC (§ 4.2): This approach targets optimistic synchronization, for exploring freshness and concurrency maximization, at the cost of consistency, possibly risking mis-calculations.

LAGOM (§ 4.3): Our balanced method for concurrent queries with low latency, "just-enough" calculation, and lightweight synchronization. Query results have stronger semantics than NOSYNC, and are based on fresher state than STRICT.

4.2 Nosync for Bulk Queries

We explore a baseline with no synchronization between updates and global queries, to explore freshness and concurrency maximization (at the cost of consistency).

F_1. To estimate F_1, improving efficiency and freshness compared to an approach as in STRICT, we introduce partial results in form of per-thread counters (T_i.F1partial) for the number of performed updates (line 2.12). The $\hat{F_1}^{\text{NOSYNC}}$ query (Eq. 1) sums these counters, improving locality, efficiency, and NUMA-friendliness compared to STRICT.

$$\hat{F_1}^{\text{NOSYNC}} = \sum_{i}^{P} T_i.\text{F1partial} \tag{1}$$

▶ **Lemma 6.** *$\hat{F_1}^{\text{NOSYNC}}$ estimates F_1 with IVL semantics.*

F_2. A global $\hat{F_2}$ equals the sum of per-partition $\hat{F_2}$, as partitions contain independent, non-overlapping parts of the input. However, an approach as for F_1 with per-partition partial results poses obstacles: threads would update the partial result at the *owning partition*, reintroducing contention and serialization on shared data into the delegation design.

Instead, *LMQ-all*$^+$ (Eq. 2) uses CM^+ to estimate F_2 for each T_i.Sk: for each key a in T_i.AF, $\mathcal{H}^{\text{NOSYNC}}$ (Eq. 3) performs a point query (Alg. 4, using the fast path on line 1.19) to obtain $\hat{f}(a)^2$, the F_2 contribution of a. However, some occurrences of a may be stored in

Algorithm 4 Point query on $T_{\mathsf{Owner}(a)}$.

```
4.1 Function pointQuery(key a)
4.2     result ← T_Owner(a).ASketchQuery(a);
4.3     foreach T_j do result += T_j.DF_Owner(a)[a];
4.4     return result;
```

Algorithm 5 $\hat{F}_2^{\text{NOSYNC}}$ on own thread.

```
5.1 Function queryF2Nosync
5.2     result ← 0;
5.3     foreach T_i do
5.4         result += F̂2^CM+(T_i.Sk);
5.5         foreach a in T_i.AF do
5.6             aFreq ← T_i.AF[a];
5.7             foreach T_j do aFreq += T_j.DF_i[a];
5.8             result += aFreq^2 − (T_i.AF^old[a])^2;
5.9     return result;
```

Algorithm 6 $\hat{F}_2^{\text{LAGOM}}$ on own thread.

```
6.1  Function queryF2Lagom
6.2      result ← 0;
6.3      foreach T_i do
6.4          T_i.beingScanned ← true;
6.5          repeat
6.6              v2 ← T_i.V2;
6.7              localResult ← min(T_i.Sk.F2partials);
6.8              foreach a in T_i.AF do
6.9                  aFreq ← T_i.AF[a];
6.10                 aFreq += P · T_i.AF^mavg[a] / 2;
6.11                 localResult += aFreq^2 − (T_i.AF^old[a])^2;
6.12             v1 ← T_i.V1;
6.13         until v1 = v2;          ▷ Retry until match
6.14         T_i.beingScanned ← false;
6.15         result += localResult;
6.16     return result;
```

$T_i.\mathsf{Sk}$ and their contribution included in the CM^+ estimate; this needs to be subtracted. We provide a geometric illustration, elaborating on the argument for this calculation, in § 5.3 and Fig. 2c. Alg. 5 shows the synchronization (or, rather, lack thereof) for concurrent *LMQ-all*$^+$.

$$LMQ\text{-}all^+ = \sum_i^P \left(\hat{F}_2{}^{CM^+}(T_i.\mathsf{Sk}) + \sum_a^{T_i.\mathsf{AF}} \mathcal{H}^{\text{NOSYNC}}(a) \right) \tag{2}$$

$$\mathcal{H}^{\text{NOSYNC}}(a) = \left(T_o.\mathsf{AF}[a] + \sum_j^P T_j.\mathsf{DF}_o[a] \right)^2 - (T_o.\mathsf{AF}^{\text{old}}[a])^2 \quad \text{where } o = \mathsf{Owner}(a) \tag{3}$$

▶ **Observation 7.** *An $\hat{F}_2{}^{\text{NOSYNC}}$ query Q can miss or double-count updates, due to data movement by overlapping processing of delegated updates (Alg. 3).*

While NOSYNC avoids the overhead of STRICT, serving as a baseline for maximal concurrency and exploration of freshness, the lack of synchronization leaves weak consistency guarantees for F_2 estimations[2], implying arbitrary fluctuations from the accuracy bounds.

4.3 Lagom for Bulk Queries – F_2

We now present our design to enable high throughput with clear concurrency semantics. We build upon some of the described components, i.e. support updates via Alg. 2, with highlighted enhancements, point queries via Alg. 4 (§ 3.11) and $\hat{F}_1$ via Eq. 1 (§ 4.2). The crux to efficient, concurrent $\hat{F}_2$ queries with consistency and accuracy guarantees lies in two key ideas: (1) *efficient maintenance of partial results*, which, based on a geometric observation, allows to argue about accuracy relative to the sequential bounds; and (2) *lightweight synchronization* implying IVL, over only few variables to scan, due to how partial results are maintained.

Partial results for F_2. While a global $\hat{F}_2$ can be obtained from summing per-partition $\hat{F}_2$ values as in $\hat{F}_2{}^{\text{NOSYNC}}$, maintaining these per partition is, unlike F_1, not straightforward. Instead, we compute partial results to simplify the heaviest operations in Eq. 2, 3, which are: (1) CM^+ on the underlying CMS, that requires reading all $H \times K$ counters of the sketch, and (2) $\mathcal{H}^{\text{NOSYNC}}$ (Eq. 3), that reads all delegation filters for each heavy key.

[2] To be precise, $\hat{F}_2{}^{\text{NOSYNC}}$ is quiescence-consistent [25]: in absence of concurrent updates, the "bad things" in Obs. 7 cannot happen. But this property is to little purpose in high-rate scenarios.

For (1), we maintain the per-row sums in Def. 1 for each sketch row, labeled F2partials in Fig. 1. This is done within the enhanced CMS update in Alg. 1, with incremental (associative) calculations. $\hat{F}_2{}^{CM^+}$ can then return the min of these H values (highlighted in Eq. 4).

For (2), we aim to avoid scanning delegation filters when estimating the frequency of heavy keys. However, as filters can buffer a significant number of updates, particularly for skewed streams, their contents should not be ignored to avoid excessive underestimation (analyzed in § 5.3). To compensate, we devise $\mathcal{H}^{\text{LAGOM}}$ (Eq. 5) as a lightweight estimation of the number of heavy-key occurrences buffered in delegation filters. For each slot, ASketch filters maintain an exponentially weighted moving average of the number of occurrences received during filter flushes (line 1.4, invoked from line 3.7). We replace the scan over delegation filters in $\mathcal{H}^{\text{NOSYNC}}$ with the following idea (highlighted in Eq. 5), requiring only one read from the ASketch filter: there are P delegation filters for a partition, projected to contain $\mathsf{AF}^{\text{mavg}}[a]$ counts of a once full and getting flushed. At any point when the query scans a partition, filters are on average half full. This yields *LMQ-proj*$^+$ (Eq. 5). In § 5 we analyze the accuracy implications and the association with the sequential bounds.

$$LMQ\text{-}proj^+ = \sum_{i}^{P} \left(\min(T_i.\mathsf{Sk.F2partials}) + \sum_{a}^{T_i.\mathsf{AF}} \mathcal{H}^{\text{LAGOM}}(a) \right) \tag{4}$$

$$\mathcal{H}^{\text{LAGOM}}(a) = \left(T_o.\mathsf{AF}[a] + P \cdot \frac{T_o.\mathsf{AF}^{\text{mavg}}[a]}{2} \right)^2 - (T_o.\mathsf{AF}^{\text{old}}[a])^2 \quad \text{where } o = \mathsf{Owner}(a) \tag{5}$$

Lightweight Synchronization. The $\hat{F}_2{}^{\text{LAGOM}}$ query (Alg. 6) uses lightweight synchronization to safely calculate *LMQ-proj*$^+$ concurrently with updates. To achieve the IVL-semantics goal (G1) set in § 3 and avoid the safety problems of $\hat{F}_2{}^{\text{NOSYNC}}$ seen in Obs. 7, $\hat{F}_2{}^{\text{LAGOM}}$ synchronizes with each partition i being scanned (i.e., with thread T_i) via a handshake, involving a pair of initially equal version numbers (as in [30]) for detecting a concurrent filter flush, and an atomic flag set by the query, to signal when the partition is being scanned.

In the common case, when T_i processes a tuple (a, v), Alg. 2 with enhancements finds available space in $T_i.\mathsf{DF}_{\mathsf{Owner}(a)}$ (condition on line 2.3); $T_i.\mathsf{DF}_{\mathsf{Owner}(a)}[a]$ is then incremented by v (line 2.6 or 2.9), but a handover is *not* triggered immediately (line 2.13 is false). $\hat{F}_2{}^{\text{LAGOM}}$ queries cannot directly account for this update before $T_i.\mathsf{DF}_{\mathsf{Owner}(a)}$ is flushed to $T_{\mathsf{Owner}(a)}$. $\hat{F}_2{}^{\text{LAGOM}}$ targets a consistent view of each partition's relevant information independently. Concurrent filter flushes (Alg. 3) are detected if version numbers mismatch (line 6.13), causing the query to retry, though only for at most one concurrent flush per partition, as subsequent flushes will be stalled by the flag (line 3.3). The sum of the collected per-partition partial results is then returned as $\hat{F}_2$. The update-query interaction implies:

▶ **Lemma 8.** *$\hat{F}_2{}^{\text{LAGOM}}$ gets an atomic snapshot per partition, and cannot deadlock with updates.*

Optimizations on Filters: Bounds and Self-Delegation. To bound the interval between filter flushes, a parameter B limits the buffering capacity of filters (line 2.7 and 2.11). Lower values for B trade performance for accuracy – both for updates, as filters are unavailable more often, and for queries, where more overlapping flushes interfere with them. The impact of B is studied in § 5.3, according to this observation, motivating tracking of updates in $\hat{F}_2{}^{\text{LAGOM}}$:

▶ **Observation 9.** *At most $r = PB$ single increment updates can be buffered in delegation filters and may thus not be explicitly observed by $\hat{F}_2{}^{\text{LAGOM}}$ queries.*

Further, in Delegation Sketch, a thread processing an update for a key it owns updates its ASketch directly. However, in LAGOM, ASketch updates require synchronizing with global queries, causing delays particularly in partitions owning heavy keys. To improve efficiency, each partition is extended with a *self-delegation filter* to buffer local updates and process them in bulk, also interfering less with queries.

4.4 Multiple Query Types – Relative Consistency

In § 3 we identify *monotonicity of scans* [17] (G2) as intuitive and desirable behavior – a query should observe the updates observed by a previously performed query. Having a single data structure implies certain monotonicity compared to a disjoint per-sketch data structure. As LMQ-Sketch consists of multiple components, an update can become visible to queries of different types at slightly different points. Here we bound how much/little "the world can differ" between query types. To this end, we identify as sub-operations of an update U the atomic operations, denoted here $U\big|_{op}$, after which U becomes visible to a query of the respective type: $U\big|_{PQ}$ at line 2.6 or 2.9, $U\big|_{F_1}$ at line 2.12, and $U\big|_{F_2}$ at line 3.11. We have $U\big|_{PQ} \longrightarrow U\big|_{F_1} \longrightarrow U\big|_{F_2}$ from program order. The cross-query consistency properties are:

▶ **Lemma 10.** *For queries $Q_1 \longrightarrow Q_2$ the monotonicity of scans for each combination follows*

	Q_2:	PQ	F_1	F_2
	PQ	*Monotonic*	*P-relaxed monotonic*	*r-relaxed monotonic*
Q_1:	F_1	*Monotonic*	*Monotonic*	*rP-relaxed monotonic*
	F_2	*Monotonic*	*Monotonic*	*Monotonic*

Note that: (1) The bounds are pessimistic, e.g. for $PQ \longrightarrow F_1$ to deviate by P updates, all must occur in the partition queried by Q_1, while for all other updates except these, i.e., where $U\big|_{F_1} \longrightarrow Q_2$, full monotonicity of scans applies. (2) P is small relative to update rates. (3) Due to the compensation scheme of $\hat{F}_2^{\text{LAGOM}}$, even in the very unlikely case that the maximum deviation in operations observed occurs, actual results differ significantly less.

5 Accuracy of F_2 Estimation

We analyze the consistency and accuracy of LMQ-Sketch for $\hat{F}_2$, as done for PQ and F_1 queries in § 3.11 and § 4.2. We develop a sequence of auxiliary designs (Table 2) and argue for their properties, ultimately arriving at $\hat{F}_2^{\text{LAGOM}}$. We develop a geometric interpretation of $\hat{F}_2$ in terms of areas of squares, illustrating accuracy properties. For each step, we describe the organization of the data structure and the F_2 query on it in a sequential setting along with its accuracy, followed by a parallelized construction and its concurrency semantics.

5.1 Count-Min Sketch & Partitioning

We begin by comparing a "*wide*" CMS with a *partitioned* one, on fixed memory budget.

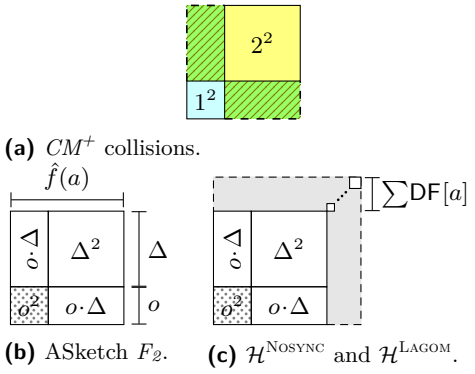
WIDE: A Count-Min Sketch with H rows and $P \times K$ columns. A toy example is shown to the side.

1		2, 3	4
4	1, 2		3

Sequential. CM^+ (Def. 1) estimates F_2 from WIDE. Consider the example CMS; each number represents a unique key hashed to that counter, and is simultaneously the frequency of that key (i.e., $f(1) = 1$, $f(2) = 2$, etc.). Colliding keys are separated by commas; CMS will only store their sum. The true F_2 is $1^2 + 2^2 + 3^2 + 4^2 = 30$. CM^+ computes

Table 2 Sequential (for accuracy reasoning) & concurrency-aware $\hat{F}_2$ (for IVL reasoning).

Data structure	Sequential	Concurrent
WIDE (CMS)	CM^+	CM^+_{conc}
PARTCMS	$PARTCMS^+$	$PARTCMS^+_{conc}$
PARTAS	$PARTAS^+$	$PARTAS^+_{conc}$
LMQ-Sketch	$LMQ\text{-}all^+$	$\hat{F}_2^{\text{NOSYNC}}$
LMQ-Sketch	$PARTAS^+$(LMQ)	$PARTAS^+_{conc}$
LMQ-Sketch	$LMQ\text{-}proj^+$	$\hat{F}_2^{\text{LAGOM}}$



(a) CM^+ collisions.

(b) ASketch F_2.

(c) $\mathcal{H}^{\text{NOSYNC}}$ and $\mathcal{H}^{\text{LAGOM}}$.

Figure 2 Geometric interpretation of $\hat{F}_2$.

$1^2 + (2+3)^2 + 4^2 = 42$ from the first row and $4^2 + (1+2)^2 + 3^2 = 34$ from the second, returning the smaller estimate. Overestimation arises from colliding keys. Fig. 2a illustrates the CM^+ calculation for the counter containing $(1, 2)$. The true F_2 contribution of the contained keys is 5, the sum of the plain areas. CM^+ calculates $(1+2)^2 = 9$, the entire square, thus *overestimating by an amount equal to the striped areas.* The impossibility of underestimating is clear. Heavy keys induce particularly large extra areas, suggesting a need to treat them specially.

Concurrent. CMS updates and queries can be parallelized by, e.g., atomic fetch-and-add. In [39], Rinberg and Keidar show a similar construction to be an IVL implementation of CMS. On top of this parallel CMS, consider CM^+_{conc} as a concurrency-aware adaptation of CM^+ using atomic reads; a similar argument as [39, Lemma 5.3] implies:

▶ **Lemma 11.** $\text{CM}^+_{\text{conc}}$ *is an IVL implementation of* CM^+*, preserving* CM^+*'s* (ϵ, δ) *bounds.*

For reduced contention relative to WIDE, the space is now partitioned into P sketches:

PARTCMS: P partitions, each with a $H \times K$ CMS. Each partition sketches a subdomain of the input. (Toy example on the right).

1	2	3, 4	
	1, 2	4	3

Sequential. Partitioning the memory of the example into $P = 2$ partitions (note each key is only present in one partition), to estimate F_2 from PARTCMS, $PARTCMS^+$ sums a CM^+ estimate for each partition. In the example, 5 is calculated for the first partition and 25 for the second. Their sum is an $\hat{F}_2$ for the complete PARTCMS.

▶ **Observation 12.** PARTCMS^+ *can only improve accuracy of* $\hat{F}_2$ *compared to* CM^+ *on* WIDE*, as it selects sketch rows with minimum overestimation independently for each partition.*

Concurrent. Each partition is assigned a thread to perform updates. *For now, assume that each thread only receives tuples for keys of its partition.* Consider $PARTCMS^+_{conc}$ as a parallelization of $PARTCMS^+$ using atomic reads (similar to CM^+_{conc}) to perform $PARTCMS^+$ concurrently with updates. Similar reasoning as in Lemma 11 implies:

▶ **Lemma 13.** $\text{PARTCMS}^+_{\text{conc}}$ *is an IVL implementation of* PARTCMS^+.

5.2 Augmented Sketch & Partitioning

ASketch [41], § 2, introduced filters to track heavy keys more accurately. In a similar fashion to PartCMS, partitioning can be applied to ASketch while keeping total memory constant:

PartAS: P ASketches, each of size $H \times K'$ – reducing width to fit filters in same total memory.

Sequential. Consider $PartAS^+$: for each partition, CM^+ is applied to its CMS; the F_2 contribution of each key a in the filters is $\hat{f}(a)^2$, shown in Fig. 2b, where $\hat{f}(a) = \mathsf{AF}[a]$ (line 1.19). Note, $\hat{f}(a)$ contains $o = \mathsf{AF}^{\text{old}}[a]$ occurrences which are also in the CMS from times when a did not reside in the filter due to not being heavy enough. Hence, CM^+ already included o^2 (dotted area in Fig. 2b). *To avoid double-counting this quantity, o^2 is subtracted* ($\mathcal{H}^{\text{Nosync}}$ (Eq. 3) and $\mathcal{H}^{\text{Lagom}}$ (Eq. 5)). Finally, the contributions are summed to get an $\hat{F}_2$.

▶ **Observation 14.** *Higher P is beneficial to* PartAS$^+$*'s accuracy, reducing overestimation through P filters accurately counting heavy keys, while maintaining one-sided error as* CM$^+$.

Concurrent. As in $PartCMS^+_{conc}$, each partition is updated by a dedicated thread, *(still) with the simplifying assumption that input tuples are distributed to the owning partition.* Instead of unsafe scanning, consider $PartAS^+_{conc}$ which obtains an atomic snapshot of each partition-local ASketch (using any synchronization that can guarantee that) and computes an estimate as $PartAS^+$, implying:

▶ **Lemma 15.** PartAS$^+_{\text{conc}}$ *is an IVL implementation of* PartAS$^+$.

5.3 Concurrency Awareness – Delegation

We now *waive the simplification of input being distributed to the owning partition*; threads delegate updates to the owner. We describe and compare three approaches to estimating F_2.

LMQ-Sketch: Each partition of PartAS uses P delegation filters to buffer updates, periodically flushed to the owning partition. K is adjusted to maintain the memory budget.

All DFs. *LMQ-all$^+$* (Eq. 2) extends $PartAS^+$ to include all delegation filters when estimating $\hat{f}(a)$, shown as $\sum\mathsf{DF}[a]$ in Fig. 2c. Buffered occurrences of light keys are not considered, as they by definition do not significantly contribute to F_2, particularly for skewed streams which we target. $\hat{F}_2{}^{\text{Nosync}}$ (§ 4.2) calculates this as a concurrent query, but lack of synchronization makes it unsafe (Obs. 7). Further, *LMQ-all$^+$* scales quadratically in P; all P delegation filters are read from other threads for each of C heavy keys in P ASketch filters.

No DFs. To query more efficiently, we consider outright ignoring delegation filters, which will underestimate F_2, by applying $PartAS^+$ to the LMQ-Sketch data structure. However, the number of buffered, hence ignored, updates per partition is bounded (Obs. 9).

▶ **Lemma 16.** PartAS$^+_{\text{conc}}$ *on LMQ-Sketch (i.e. its ASketch part) is an r-relaxed IVL implementation of* LMQ-all$^+$ *per partition, where $r = PB$.*

▶ **Corollary 17.** *Since IVL is a local property [39],* PartAS$^+_{\text{conc}}$ *on LMQ-Sketch is an rP-relaxed IVL implementation of* LMQ-all$^+$.

Note the worst-case is extremely unlikely to happen, as all r updates would need to be for the same key (i.e. extreme skew), and filters do not become full simultaneously.

Lagom. To compensate for this relaxation yet maintain its high efficiency, *LMQ-proj*$^+$ (§ 4.3) projects the number of buffered occurrences of heavy keys to replace the exact calculation of $\sum \mathsf{DF}[a]$ in Fig. 2c and $\mathcal{H}^{\text{Nosync}}$ (Eq. 3).

▶ **Observation 18.** *$\mathcal{H}^{\text{Lagom}}$ (Eq. 5) projection compensates for up to $r/2$ ignored occurrences per heavy key, bringing the query result closer to the actual F_2, based on the reasoning of the geometric argument and observations 12 and 14.*

From the above and Lemma 8, we have:

▶ **Lemma 19.** *$\hat{F}_2{}^{\text{Lagom}}$ is an IVL implementation of* LMQ-proj$^+$.

The behavior of $\hat{F}_2{}^{\text{Lagom}}$ within the bounds is data- and execution-dependent and is evaluated in the next section. Summarizing the properties of LMQ-Sketch, we have:

▶ **Corollary 20.** *LMQ-Sketch is a composite concurrent sketch data structure for multiple queries: (1) PQ (§ 3.11), an IVL implementation of the ASketch point query; (Lemma 5); (2) $\hat{F}_1{}^{\text{Nosync}}$, an IVL implementation of exact F_1 (Lemma 6); (3) $\hat{F}_2{}^{\text{Lagom}}$, an IVL implementation of* LMQ-proj$^+$ *(Obs. 18, Lemma 19). Queries follow the monotonicity properties in Lemma 10.*

6 Evaluation

Baselines. We study LMQ-Sketch relative to: (1) SW-SKT [9], the software implementation of SKT which targets FPGA-acceleration for high-rate sketching. SKT uses separate sketches for multiple queries (F_0, point queries, and F_2 via HLL, CMS, and Fast-AGMS) updated in parallel, but *no concurrent queries* (first merging all thread-local sketches before querying). This comparison is for insights about answering mixed queries in a single sketch, from a memory, accuracy and scalability point of view. (2) Delegation Sketch [42], which supports concurrent updates but *with point queries only*. By comparing, we aim to understand the synchronization overhead for concurrent global queries. (3) The elementary Strict and Nosync designs are used to evaluate the balance achieved by Lagom. Note that simply adding concurrent queries to SW-SKT would imply effects similar to Nosync and Strict and hence are not studied separately.

Datasets and Hardware Platform. From the CAIDA [6] network packet traces we extract 18.5 M tuples of headers. Besides, synthetic datasets with skew $z = 1$, 1.5, 2 and 3 (each having 100 M tuples sampled from a domain of size 1 M) are used to explore performance characteristics up to extreme skew levels where a majority of input tuples are owned by a single partition. All experiments are conducted on a 128-core AMD EPYC 7954 running at 2.25 GHz, without using SMT. The server runs openSUSE Tumbleweed 20250211. LMQ-Sketch is implemented in C++23 [26], compiled with GCC 14.2.1 and `-O3 -march=native`.

Metrics. To study the scalability and efficiency of LMQ-Sketch, we measure *throughput* (updates per unit-time) at different thread counts, without and with concurrent queries. To study accuracy in presence of concurrency, we focus on *latency* of global queries, which has implications on the IVL-permitted interval; we propose a methodology to evaluate *accuracy of IVL-associated queries* (cf. § 5), measuring the difference of returned values relative to the estimated IVL-bounds, giving insight into the synergy of IVL semantics, query latency and freshness. Finally, we compare the impact of memory budget on accuracy for $\hat{F}_2{}^{\text{Lagom}}$, state-of-the-art sketching techniques, and reference methods in § 5.

Parameters. *Threads, partitions & memory*: we stress-test with high thread counts ($P = 1$ to 128). Memory budget per partition is constant at 32 KiB: $H{\times}K = 8{\times}1024$, while reducing K when having filters (cf. evaluations in [41, 42]), and keeping sketch rows (hash function computations) fixed. *Filter size*: C is kept at 16 as in [42], delegation filter buffer capacity is bounded at $B = 1000$. *Synchronization*: we compare Lagom with the aforementioned baselines, including two Delegation Sketch configurations: the plain one (unbounded B) and one with $B{=}1000$. *Skewness of input data*: $z > 1$ for the synthetic data, and $z \approx 1.4$ for the CAIDA data. *Frequency of concurrent queries*: stress testing with global queries at rates of 1000/s and point queries for 0.1 % of update tuples, following the benchmark in [42].

Experiments. We begin by comparing the impact of synchronization on update operation's throughput when processing a complete input dataset from system memory (§ 6.1). Next, we investigate the accuracy and freshness of global query results, in relation to memory budget and concurrency. Higher weight lies on the more complicated F_2 queries, $\hat{F}_1$'s behavior being studied mainly in terms of query latency and PQ's behavior evaluated in [42] (§ 6.2, 6.3, 6.4). For brevity, the results are shown in figures with summary descriptions in the caption, and main takeaways in associated paragraphs. More detailed discussions appear in Appendix B.

6.1 Concurrency and Synchronization

Design and Parameters. Each dataset is processed with varying numbers of partitions and threads, P. First, we measure update throughput with no concurrent queries. We then repeat the measurement with a constant high load of concurrent queries: F_1 and F_2 at 1000/s each, and point queries at 0.1 % of updates, for the designs that support them.

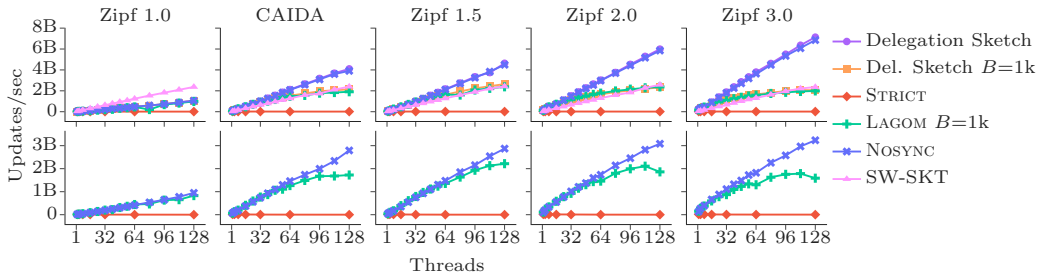


Figure 3 Mean update throughput, without (upper row) and with concurrent queries (lower row, for designs that support them). Fluctuations are small and omitted for clarity. Note the minimal overhead of Lagom for global queries as it matches Delegation Sketch with B=1k. Strict does not scale with increasing threads or skew. With concurrent queries, Lagom stays close to Nosync, affirming the lightweight-ness of the synchronization design; increasing skew improves performance, as the latency hiding of delegation counteracts and postpones saturation even at extreme skew.

Takeaways. The results, shown in Fig. 3, imply that LMQ-Sketch with Lagom is able to maintain state in a way that allows independence by different threads, enabling high processing throughput, yet still imparting the necessary consistency to serve the purpose of sketching for continuously and concurrently answering multiples queries in a streaming setting.

6.2 Memory and Accuracy

We consider a sequential setting, to explore the impact of sketch memory budget on F_2 estimation accuracy for several baselines and our methods. *Note*: There are *two main parameters for memory budget*, but with differing impact on accuracy: *per-partition memory* (given by H and K) and the *total number of partitions* P (also number of threads when parallelizing). In the partitioned design, an increase in any of these improves accuracy (§ 5).

Design and Parameters. The methods in the analysis (§ 5) form points of reference; additionally, we compare with Fast-AGMS [10], used in SW-SKT. For each method and skew $z = 1$, 1.5 and 2, five synthetic datasets (length 100M, cardinality 1M) are processed under various memory budgets, for $\hat{F}_2$ and the mean absolute percent error (MAPE) relative to the true F_2, at the end of the execution (i.e., based on the global state for the filter-enabled designs). For each method in § 5, we use constant per-partition memory 32 KiB ($H \times K = 8 \times 1024$ with adjustments in presence of filters, as before); total memory increases with P.

Fig. 4 shows the MAPE in log-scale, along the outcome for a single-partition Fast-AGMS at several budgets: 6×2^{13} (as in the original evaluation of [9]), 8×1024 (same $P = 1$ in the partitioned approaches), and $128 \times 8 \times 1024$ (identical to the total budget of LMQ-Sketch at $P = 128$). Increasing parallelism for thread-local designs such as SW-SKT will require P times this memory at runtime, but the accuracy bounds of queries remain the same as for a single sketch of $1/P$ of the total memory (since the local sketches simply get merged). Fig. 5 shows memory budget scaling for SW-SKT and LMQ-Sketch when the number of partitions/local-sketches follow the number of threads when run in parallel.

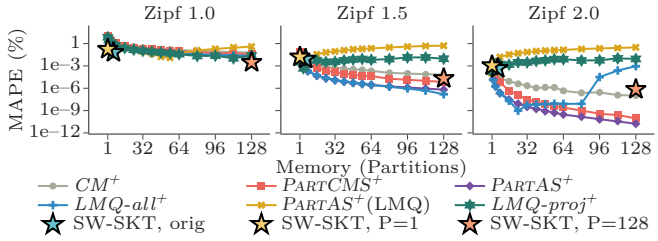


Figure 4 Non-concurrent MAPE for F_2 estimations at various z and memory budgets, in multiples of 1 partition occupying $H \times K = 8 \times 1024$ counters (32 KiB). Thread-local approaches (SW-SKT) must merge local sketches before querying, and achieve the accuracy of this single sketch, various sizes of which are shown here. For such designs, increasing memory budget with additional threads does not improve accuracy.

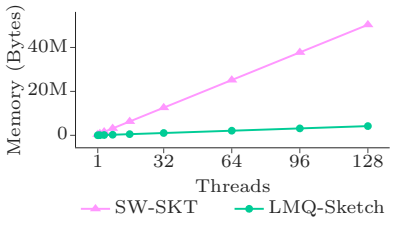


Figure 5 Total memory scaling linearly with P. LMQ-Sketch uses 32 KiB per thread/partition. SW-SKT, with 6×2^{13} counters for its CMS and Fast-AGMS as in the evaluation of [9] (but without the HLL for F_0) uses 393 kB per thread.

Takeaways. *PartCMS*$^+$ and *PartAS*$^+$, our stepwise enhancements for *CM*$^+$ in § 5, improve estimation accuracy. Unlike for thread-local designs, the partitioned approach allows LMQ-Sketch to utilize the increased memory budget for a *two-fold benefit*: increasing per-partition memory improves (sequential) accuracy in isolation; increasing the number of partitions benefits both accuracy and parallelism. Although Fast-AGMS is one of the most accurate F_2 sketches, its thread-local-oriented design does not see improved accuracy for the same per-partition memory budget, while LMQ-Sketch achieves the same or higher accuracy by effectively navigating the memory and concurrency trade-offs.

6.3 Query Latency and Accuracy

Next, we consider concurrent queries. As described in (G1) (§ 3), IVL, while preserving bounds for sketches, cannot fully characterize the final accuracy of the result; query latency plays a key role. LMQ-Sketch addresses this by optimizing query latency using partial results at insertion. We evaluate the latency improvement due to this algorithmic design.

Design and Parameters. We measure the latency of global queries with different values of P, concurrent with updates. After 100 ms of warmup to populate delegation filters, queries are performed at a rate of 1000/s. For F_1 queries, we focus on the $\hat{F}_1{}^{\text{Nosync}}$ IVL design (§ 4.2). For F_2, the more complex query, we benchmark the synchronization designs.

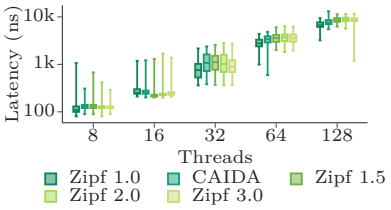


Figure 6 Latency of $\hat{F}_1{}^{\text{Nosync}}$ queries concurrent with updates scales linearly with P. No impact from skew.

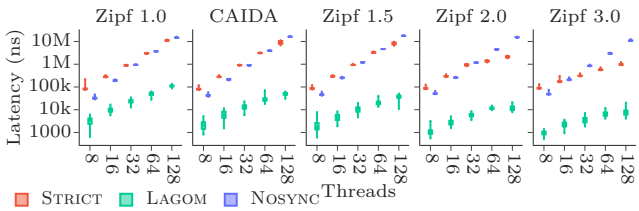


Figure 7 Latencies of 100 F_2 queries concurrent with updates. Lagom is 2-3 orders of magnitude faster than other designs, which struggle with high thread counts.

Takeaways. $\hat{F}_1{}^{\text{Nosync}}$ query duration ranges from 100 ns to 10 µs at high thread counts. Our optimizations in $\hat{F}_2{}^{\text{Lagom}}$ yield significant improvements in query latency, which stays below 100 µs. Such short latencies suggest the suitability of our queries for accurate estimation under IVL, due to reduced number of overlapping updates, evaluated next.

6.4 Concurrency and Accuracy

Building on insights from measuring concurrent query latencies, we now evaluate the impact on accuracy. Query duration in conjunction with update throughput determine the IVL interval size. Although IVL does not exactly target accuracy guarantees, to complement, we explore the admissible freshness of this interval.

Design and Parameters. We compare the return value of concurrent $\hat{F}_2{}^{\text{Lagom}}$ with the IVL-permitted interval. We adopt the following methodology to determine the relative error induced by concurrent executions, without interfering with execution patterns, e.g., as an approach based on stopping and starting updater threads to record measurements would do.

Methodology to determine bounds of return interval of concurrent query Q: We reconstruct ideal query return values at Q^{start} and Q^{end}. A threshold F_1-value T is selected (beyond the warmup period), as a trigger point for performing Q which overlaps an arbitrary number of updates n. To reconstruct Q^{start}, updates are stopped when $F_1 = T$ and a sequential query is performed. In a new execution, reaching $F_1 = T$ instead triggers Q concurrently. Upon the completion of Q, updates are stopped and Q^{end} is recorded in a sequential setting.

We set $T = 10$M tuples and perform 50 repetitions for each dataset (CAIDA and synthetic with $z = 1$, 1.5 and 2) and increasing P. We study $\hat{F}_2{}^{\text{Nosync}}$ and $\hat{F}_2{}^{\text{Lagom}}$ using the same methodology to determine the effect of latency and weaker semantics on accuracy.

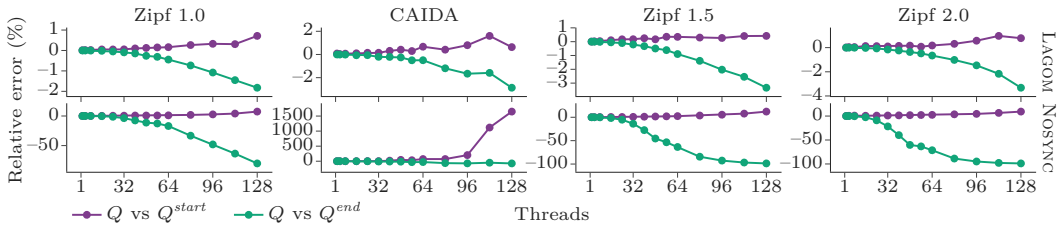


Figure 8 Comparison of query return value against bounds of IVL-permitted interval. For both designs, Q observes more updates than Q^{start} (which is what $\hat{F}_2^{\text{STRICT}}$ would return) but misses some updates observed by Q^{end}. LAGOM consistently yields (orders of magnitude) narrower return value intervals compared to NOSYNC, which may miss many updates due to its long operation latency.

Takeaways. The concurrent F_2 queries can observe overlapping updates (towards improving STRICT's freshness). However, the interval of return values for NOSYNC is arbitrarily large, while LAGOM (with agile calculation and IVL guarantees) imparts a seriously smaller uncertainty, particularly at low–intermediate P, demonstrating that LAGOM's properties preserve, and are useful for, accuracy.

7 Other Related Work

Rinberg et al. in [40] present a snapshot-based methodology for sketch *querying concurrently with updates*, building on a thread-local design with limited local buffering and a global propagator which merges local sketches into a global one when full. Concurrent queries require a snapshot of this global sketch, which gets more costly with the size of the state, also leading to a tradeoff with respect to accuracy. Using a partitioning design as in LMQ-Sketch allows efficient queries taking snapshots per partition, while supporting distributed state and fine-tuning of freshness. Concurrent queries and updates are also targeted in [18] for estimating quantiles, although not using the generic framework of [40] due to risk of sequential bottlenecks.

Several recent orthogonal works such as Hydra [32] and OmniSketch [38] explore using sketches for *multiple querying* of multidimensional data streams. These works are not targeting queries concurrent with updates, though, instead using an online sketching phase followed by an offline querying phase, during which the sketch is no longer updated.

Universal Sketching appears in [31, 32] as a useful technique to estimate sums of monotonic functions applied to stream frequency vectors, such as F_1 and F_2, from a single sketch in small (polylogarithmic) memory. As its core functions associate with the ones in LMQ-Sketch, the latter can be a possible candidate for supporting concurrency in it.

8 Conclusions

LMQ-Sketch is a concurrent, multi-query sketch in a single, low memory-footprint object, with explicit concurrency semantics that lead to predictably high accuracy even with very demanding stream properties. The analytical and detailed empirical insights using real-world and synthetic data showed that (1) having a single data structure balances multiple targets: accuracy (Fig. 4), timeliness (Fig. 6, 7), memory footprint (Fig. 5), freshness (Fig. 8), concurrency, and throughput (Fig. 3); (2) besides the memory budget, the way that the memory is used is catalytic for the achievable concurrency and accuracy, both for local and

global queries; (3) employing IVL, in conjunction with low query latency and concurrency-aware compensation, enables accuracy aligning with that of the sequential sketches, even under very high-rate, skewed streams.

LMQ-Sketch can be a useful component in systems such as Redis, Apache Druid/Spark/DataSketches and more. The method facilitates continued research towards supporting additional queries concurrently, which all need to deal with the same challenges regarding efficiency, consistency, and accuracy; e.g. top-k elements, possibly supporting wavelets, quantiles, and associating with queries required for universal sketching.

References

1 Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '99, pages 10–20, New York, NY, USA, May 1999. Association for Computing Machinery. doi:10.1145/303976.303978.

2 DataSketches | Apache® DataSketches. URL: https://datasketches.apache.org/.

3 Apache Druid | Apache® Druid. URL: https://druid.apache.org/.

4 Vladimir Braverman and Rafail Ostrovsky. Zero-one frequency laws. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, pages 281–290, New York, NY, USA, June 2010. Association for Computing Machinery. doi:10.1145/1806689.1806729.

5 Chiranjeeb Buragohain and Subhash Suri. Quantiles on Streams. In *Encyclopedia of Database Systems*, pages 2235–2240. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_290.

6 The CAIDA UCSD Anonymized Internet Traces - 2018. URL: https://www.caida.org/catalog/datasets/passive_dataset.

7 Lidia Ceriani and Paolo Verme. The origins of the Gini index: Extracts from Variabilità e Mutabilità (1912) by Corrado Gini. *The Journal of Economic Inequality*, 10(3):421–443, September 2012. doi:10.1007/s10888-011-9188-x.

8 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding Frequent Items in Data Streams. In *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 693–703, Berlin, Heidelberg, 2002. Springer. doi:10.1007/3-540-45465-9_59.

9 Monica Chiosa, Thomas B. Preußer, and Gustavo Alonso. SKT: A one-pass multi-sketch data analytics accelerator. *Proceedings of the VLDB Endowment*, 14(11):2369–2382, July 2021. doi:10.14778/3476249.3476287.

10 Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 13–24, Trondheim, Norway, August 2005. VLDB Endowment. URL: http://www.vldb.org/archives/website/2005/program/paper/tue/p13-cormode.pdf.

11 Graham Cormode and Minos Garofalakis. Join sizes, frequency moments, and applications. In *Data Stream Management: Processing High-Speed Data Streams*, pages 87–102. Springer, 2016. doi:10.1007/978-3-540-28608-0_4.

12 Graham Cormode, Minos Garofalakis, and Dimitris Sacharidis. Fast Approximate Wavelet Tracking on Streams. In *Advances in Database Technology - EDBT 2006*, pages 4–22, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11687238_4.

13 Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, April 2005. doi:10.1016/j.jalgor.2003.12.001.

14 Graham Cormode and S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 44–55. Society for Industrial and Applied Mathematics, April 2005. doi:10.1137/1.9781611972757.5.

15 Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge University Press, Cambridge, 2020. doi:10.1017/9781108769938.

16 Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Sketch-Based Multi-Query Processing over Data Streams. In *Data Stream Management: Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pages 241–261. Springer, Berlin, Heidelberg, 2016. doi:10.1007/978-3-540-28608-0_12.

17 Cynthia Dwork, Maurice Herlihy, Serge Plotkin, and Orli Waarts. Time-Lapse Snapshots. *SIAM Journal on Computing*, 28(5):1848–1874, January 1999. doi:10.1137/S0097539793243685.

18 Shaked Elias Zada, Arik Rinberg, and Idit Keidar. Quancurrent: A Concurrent Quantiles Sketch. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 15–25, New York, NY, USA, June 2023. Association for Computing Machinery. doi:10.1145/3558481.3591074.

19 Minos Garofalakis. Discrete Wavelet Transform and Wavelet Synopses. In *Encyclopedia of Database Systems*, pages 857–863. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_539.

20 Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 454–465, Hong Kong, China, August 2002. VLDB Endowment. doi:10.1016/B978-155860869-6/50047-0.

21 Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. One-pass wavelet decompositions of data streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):541–554, May 2003. doi:10.1109/TKDE.2003.1198389.

22 Corrado Gini. *Variabilità e Mutabilità*. Reprinted in Memorie Di Metodologia Statistica (Ed. E. Pizetti and T. Salvemini.) 1955. Libreria Eredi Virgilio Veschi, Rome, 1912.

23 Itamar Haber. Count-Min Sketch: The Art and Science of Estimating Stuff, March 2022. URL: https://redis.io/blog/count-min-sketch-the-art-and-science-of-estimating-stuff/.

24 Thomas A. Henzinger, Christoph M. Kirsch, Hannes Payer, Ali Sezgin, and Ana Sokolova. Quantitative relaxation of concurrent data structures. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 317–328, New York, NY, USA, January 2013. Association for Computing Machinery. doi:10.1145/2429069.2429109.

25 Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. *The Art of Multiprocessor Programming*. Elsevier, Morgan Kaufmann Publishers, Cambridge, MA, United States, second edition, 2021.

26 Martin Hilgendorf. LMQ-Sketch. URL: https://gitlab.com/marhilg/lmq-sketch.

27 Victor Jarlow, Charalampos Stylianopoulos, and Marina Papatriantafilou. QPOPSS: Query and Parallelism Optimized Space-Saving for finding frequent stream elements. *Journal of Parallel and Distributed Computing*, 204:105134, October 2025. doi:10.1016/j.jpdc.2025.105134.

28 Lior Kogan. T-digest: A New Probabilistic Data Structure in Redis Stack, March 2023. URL: https://redis.io/blog/t-digest-in-redis-stack/.

29 Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 234–247, New York, NY, USA, October 2003. Association for Computing Machinery. doi:10.1145/948205.948236.

30 Leslie Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, November 1977. doi:10.1145/359863.359878.

31 Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, Florianopolis Brazil, August 2016. ACM. doi:10.1145/2934872.2934906.

32 Antonis Manousis, Zhuo Cheng, Ran Ben Basat, Zaoxing Liu, and Vyas Sekar. Enabling efficient and general subpopulation analytics in multidimensional data streams. *Proceedings of the VLDB Endowment*, 15(11):3249–3262, July 2022. doi:10.14778/3551793.3551867.

33 Gonçalo Matos, Salvatore Signorello, and Fernando M. V. Ramos. Generic change detection (almost entirely) in the dataplane. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ANCS '21, pages 113–120, New York, NY, USA, January 2022. Association for Computing Machinery. doi:10.1145/3493425.3502767.

34 Vinh Quang Ngo and Marina Papatriantafilou. Cuckoo heavy keeper and the balancing act of maintaining heavy hitters in stream processing. *Proceedings of the VLDB Endowment*, 18(9):3149–3161, August 2025. doi:10.14778/3746405.3746434.

35 Yiannis Nikolakopoulos, Anders Gidenstam, Marina Papatriantafilou, and Philippas Tsigas. A Consistency Framework for Iteration Operations in Concurrent Data Structures. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 239–248, May 2015. doi:10.1109/IPDPS.2015.84.

36 Savannah Norem. Probabilistic Data Structures in Redis, August 2022. URL: https://redis.io/blog/streaming-analytics-with-probabilistic-data-structures/.

37 Erez Petrank and Shahar Timnat. Lock-Free Data-Structure Iterators. In *Proceedings of the 27th International Symposium on Distributed Computing - Volume 8205*, DISC 2013, pages 224–238, Berlin, Heidelberg, October 2013. Springer-Verlag. doi:10.1007/978-3-642-41527-2_16.

38 Wieger R. Punter, Odysseas Papapetrou, and Minos Garofalakis. OmniSketch: Efficient Multi-Dimensional High-Velocity Stream Analytics with Arbitrary Predicates. *Proc. VLDB Endow.*, 17(3):319–331, November 2023. doi:10.14778/3632093.3632098.

39 Arik Rinberg and Idit Keidar. Intermediate Value Linearizability: A Quantitative Correctness Criterion. *Journal of the ACM*, 70(2):17:1–17:21, April 2023. doi:10.1145/3584699.

40 Arik Rinberg, Alexander Spiegelman, Edward Bortnikov, Eshcar Hillel, Idit Keidar, Lee Rhodes, and Hadar Serviansky. Fast Concurrent Data Sketches. *ACM Transactions on Parallel Computing*, 9(2):6:1–6:35, April 2022. doi:10.1145/3512758.

41 Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented Sketch: Faster and More Accurate Stream Processing. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1449–1463, New York, NY, USA, June 2016. Association for Computing Machinery. doi:10.1145/2882903.2882948.

42 Charalampos Stylianopoulos, Ivan Walulya, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. Delegation sketch: A parallel design with support for fast and accurate concurrent operations. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, pages 1–16, New York, NY, USA, April 2020. Association for Computing Machinery. doi:10.1145/3342195.3387542.

43 Da Tong and Viktor K. Prasanna. Sketch Acceleration on FPGA and its Applications in Network Anomaly Detection. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):929–942, April 2018. doi:10.1109/TPDS.2017.2766633.

44 R. Kent Treiber. *Systems Programming: Coping with Parallelism.* Number 5118 in Research Report. International Business Machines Incorporated Research Division, Thomas J. Watson Research Center, 1986. URL: https://dominoweb.draco.res.ibm.com/reports/rj5118.pdf.

45 David P. Woodruff. Sketching as a Tool for Numerical Linear Algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, October 2014. doi:10.1561/0400000060.

46 David P. Woodruff. Frequency Moments. In *Encyclopedia of Database Systems*, pages 1518–1519. Springer, New York, NY, 2018. doi:10.1007/978-1-4614-8265-9_167.

A Proof arguments

Due to space limitations, arguments that support the claims in the main part of the paper are presented in more detail in this appendix.

A.1 From § 4

▶ **Lemma 5.** *Delegation Sketch-based PQ is an IVL implementation of ASketch point query.*

Proof sketch. While $T_{\mathsf{Owner}(a)}$ executes PQ, other threads may concurrently update relevant delegation filters. However, only updates overlapping PQ can be missed; updates completed before PQ are reflected in the returned $\hat{f}(a)$ [42, Claim 2]. Double-counting of updates is not possible [42, Claim 3], as this would mean PQ observed an update both while it is buffered in a delegation filter, as well as when it has been flushed to the partition-local ASketch. This would require a flush of the delegation filter, which can only be performed by the same thread as PQ itself, and hence cannot overlap PQ. Therefore, the maximal return value is the one for a linearization which observes all overlapping updates. ◀

▶ **Lemma 6.** $\hat{F}_1{}^{\textsc{Nosync}}$ *estimates* F_1 *with IVL semantics.*

Proof sketch. The partial results form jointly a shared counter, where each thread maintains a local counter of completed updates. The query, performing one atomic read per partition, can neither double-count updates not omit completed ones. ◀

▶ **Observation 7.** *An* $\hat{F}_2{}^{\textsc{Nosync}}$ *query* Q *can miss or double-count updates, due to data movement by overlapping processing of delegated updates (Alg. 3).*

This can occur when Q overlaps with processing delegated updates (Alg. 3), which non-atomically moves occurrences of keys from delegation filters to the ASketch of the owning partition. Updates can be missed when Q scans a sketch $T_i.\mathsf{Sk}$ (line 5.4), whereupon a filter $T_j.\mathsf{DF}_i$ is handed over to T_i and flushed (Alg. 3), followed by Q reading the now-empty filter (line 5.7), thus missing all the concurrently flushed updates, regardless of whether they were completed before Q began. Or, if Q overlaps a flush of $T_j.\mathsf{DF}_i$, updates may be seen both in the ASketch of T_i and in $T_j.\mathsf{DF}_i$ before the latter is cleared.[3]

▶ **Lemma 8.** $\hat{F}_2{}^{\textsc{Lagom}}$ *gets an atomic snapshot per partition, and cannot deadlock with updates.*

Proof sketch. *Atomic per-partition snapshot* – When a full delegation filter is flushed, buffered updates move to the owning partition's ASketch and associated partial results are updated (Alg. 1). As this operation is not atomic, the query synchronizes with it to avoid mis-calculations, such as double-counting updates or omitting completed ones:

- A flush begins by incrementing $\mathsf{V1}$ and increments $\mathsf{V2}$ upon completion (line 3.4 and 3.11). A query reads these in reverse order ($\mathsf{V2}$ then $\mathsf{V1}$) to detect concurrent modification of the data it read, and the scan is retried is they do not match (condition on line 6.13).
- To prevent unbounded retrying in the (unlikely) event that repeated flushes occur in a query's duration, the query flags the partition it is scanning (line 6.4). If the flush finds the flag set (line 3.3), it waits until the query has cleared the flag (line 6.14).
- If the flag is not set, the flush can proceed to increment $\mathsf{V1}$ (line 3.4) and begin updating the ASketch contents. If a query now sets the flag, it will either find the version numbers match and knows that a consistent view was obtained, or detect a mismatch, indicating the observed state may be inconsistent, and retry the scan (line 6.13). The query can be blocked by *at most one* concurrent filter flush, as subsequent ones will be stalled by the flag.

[3] Similarly, there can there be omissions and double-counting if we swap the relative order of scanning of sketches and delegation filters.

The synchronization design performs a double-collect of the version numbers (line 6.13) which indicate concurrent modification of the ASketch of the partition being scanned. As a scan gets an image of a partition's partial results without interferences from filter flushes, it sees a linearizable view of that information in the partition.

No deadlock – Deadlock entails one or more updater threads and the F_2 query thread being unable to proceed because they are waiting for one another. The query algorithm interacts with only one partition at a time; all remaining updater threads for other partitions are not involved and continue executing independently. If the query has set the flag for updater thread T_i (line 6.4), T_i will not proceed past line 3.3. Within bounded steps, the query will complete its scan of partition i, as T_i is currently blocked and cannot update the version numbers to mismatch, and ultimately reset the flag (line 6.14), thus unblocking T_i. Similarly, if the query is attempting obtain a consistent scan while T_j is performing a filter flush operation and has incremented V1 – causing the query to retry –, from our system properties, T_i is guaranteed to make progress and complete the flush within bounded time, finally incrementing V2 to match. Again, the query will be blocked by *at most one* concurrent filter flush, as subsequent ones will be stalled by the flag. ◀

▶ **Lemma 10.** *For queries $Q_1 \longrightarrow Q_2$ the monotonicity of scans for each combination follows*

	Q_2:	PQ	F_1	F_2
	PQ	*Monotonic*	*P-relaxed monotonic*	*r-relaxed monotonic*
Q_1:	F_1	*Monotonic*	*Monotonic*	*rP-relaxed monotonic*
	F_2	*Monotonic*	*Monotonic*	*Monotonic*

Proof sketch. If Q_1 and Q_2 are of the same type or if $U\big|_{Q_2} \longrightarrow U\big|_{Q_1}$ according to program order, then monotonicity of scans is immediate. There are three remaining cases where it is possible that $U\big|_{Q_1} \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow U\big|_{Q_2}$ when U overlaps both queries:

PQ* $\longrightarrow$ *F_1 At most one update operation per updater thread can overlap both Q_1 and Q_2. Therefore, a PQ is limited to observing at most P updates that are not yet visible to a subsequent F_1 query.

PQ* $\longrightarrow$ *F_2 Since Q_1 is a local query, hence only observes one partition, at most $r = PB$ updates buffered in delegation filters for this partition may be observed by Q_1 but not by Q_2 (Obs. 9).

F_1* $\longrightarrow$ *F_2 Similar to the previous case; however, Q_1 is now a global query and may in the worst case observe up to $r = PB$ buffered updates *per partition* not yet visible to Q_2. Hence, the queries may deviate from each other by up to rP updates, but no more. ◀

A.2 From § 5

▶ **Lemma 11.** $\mathrm{CM}^{+}_{\mathrm{conc}}$ *is an IVL implementation of* CM^{+}*, preserving* CM^{+}*'s* (ϵ, δ) *bounds.*

Proof sketch. Following a similar argument as [39, Lemma 5.3]; as updates only increment counters, each counter read by a CM^{+}_{conc} query Q will return a value at least as large as the value at the start of Q, and no larger than the value at the end of Q. Transitively, this holds also for the sum of squared values of said counters as computed by CM^{+}_{conc}. The returned $\hat{F}_2$ will be the minimum of these per-row results, and it cannot be lower than the least per-row result at the start of the query, or larger than the least per-row result at the end of the query, as required for IVL of monotonically increasing quantities (Obs. 2). ◀

▶ **Lemma 13.** $\textsc{PartCMS}^{+}_{\mathrm{conc}}$ *is an IVL implementation of* $\textsc{PartCMS}^{+}$.

Proof sketch. Similar reasoning as for Lemma 11. ◀

▶ **Lemma 15.** PartAS$^{+}_{\text{conc}}$ *is an IVL implementation of* PartAS^{+}.

Proof sketch. Similar to earlier lemmas. By virtue of being atomic, the snapshots for each partition will linearize with updates by the corresponding updater thread. Since counter values read by the query are non-decreasing, the return value is bounded between an ideal return value at the start of the query (observing all preceding updates) and an ideal return value at the end of the query (additionally observing all concurrent updates). ◀

▶ **Lemma 16.** PartAS$^{+}_{\text{conc}}$ *on LMQ-Sketch (i.e. its ASketch part) is an r-relaxed IVL implementation of* LMQ-all^{+} *per partition, where* $r = PB$.

Proof sketch. For a partition, up to $r = PB$ updates buffered in delegation filters (Obs. 9) may be missing from the atomic snapshot taken by $PartAS^{+}_{conc}$. Hence, the return value is bounded between the value at query start excluding the r buffered updates, and the value at the end, including all r buffered updates. ◀

B Evaluation & Discussion

Due to space limitations, the discussion supporting the main takeaways of the empirical study is presented in more detail in this appendix.

B.1 Concurrency and Synchronization (§ 6.1) – Results

Fig. 3 shows the mean rate of update operations for sketching the complete input data sets. Lagom exhibits very similar scaling as Delegation Sketch with $B = 1000$, processing approximately 1.5 billion updates/second on real-world data, demonstrating the very low overhead of its synchronization design for consistent global queries. Strict cannot scale and performance worsens with more threads as contention around the global RW lock increases. Nosync scales similarly to the plain Delegation Sketch (neither of them support consistent global queries). As expected, SW-SKT throughput grows linearly with the number of threads, as threads process updates entirely independently. However, concurrent queries cannot be supported. With increasing skewness, a clear upwards trend in throughput of the delegation design is seen, as delegation filters are able to buffer more updates locally, reducing inter-thread communication.

On the lower row, Fig. 3 shows the mean rate of update operations with concurrent queries for designs which support them. Strict shows similar scaling to before. Lagom achieves very similar performance in presence of concurrent queries. Decreases in update throughput are explained by the fact that updater threads are responsible for serving point queries alongside their update workload. Nosync clearly shows this effect as global queries have no impact in this design, but the overhead of point query work leads to a reduction in peak throughput not seen in Lagom by around 50 % compared to Fig. 3 at 0.1 % point queries (exactly reproducing the result in [42]).

B.2 Memory and Accuracy (§ 6.2) – Results

Fig. 4 shows the accuracy of various F_2 methods, without concurrent updates during query execution. As discussed in § 5, $PartCMS^{+}$ and $PartAS^{+}$, our enhanced versions of CM^{+} for more efficient data structures, improve estimation accuracy. Further, for partitioning-based approaches, increasing the memory budget by increasing the number of partitions leads to improved accuracy, as more memory is available for (1) avoiding hash collisions (Obs. 12) and

(2) accurate tracking of heavy keys in ASketch filters (Obs. 14). Although Fast-AGMS, used in SW-SKT, is one of the most accurate techniques for F_2 estimation for arbitrary skewness, a thread-local design does not see improved accuracy for the same memory budget, while LMQ-Sketch's partition-targeting methods, navigating memory and concurrency trade-off challenges, achieve higher accuracy. Of course, to be fair, one should observe that SW-SKT was not designed with concurrent queries as a target.

B.3 Latency and Accuracy (§ 6.3) – Results

Fig. 6 shows $\hat{F}_1{}^{\text{NOSYNC}}$ query latency to be small – implying a low number of overlapping updates – growing linearly with the number of threads and partitions and not impacted by skew.

Similarly, Fig. 7 shows the distribution of F_2 query durations for various synchronization designs. The operational complexity of $\hat{F}_2{}^{\text{NOSYNC}}$ scales with the number of delegation filters and ASketch filters, as for each slot in ASketch filters, all delegation filters of that partition are read ($\mathcal{H}^{\text{NOSYNC}}$, Eq. 3). $\hat{F}_2{}^{\text{LAGOM}}$ is significantly faster, both absolutely and comparatively, at less than 100 µs, 2 to 3 orders of magnitude faster than $\hat{F}_2{}^{\text{STRICT}}$ and $\hat{F}_2{}^{\text{NOSYNC}}$, regardless of skew, demonstrating the impact of our projection-based latency optimization.

B.4 Concurrency and Accuracy (§ 6.4) – Results

Fig. 8 shows how query return values relate to IVL interval boundaries for $\hat{F}_2{}^{\text{LAGOM}}$ and $\hat{F}_2{}^{\text{NOSYNC}}$. In all cases, the concurrent query Q observes more updates than Q^{start} – which is what $\hat{F}_2{}^{\text{STRICT}}$ would return – but ignores some updates that Q^{end} observes, as expected. Thus, the truly concurrent $\hat{F}_2$ query results are more fresh than STRICT synchronization. However, the size of the interval for NOSYNC is large compared to LAGOM, due to its significantly longer query latency (seen in Fig. 7) and the weakness of its semantics (described in Obs. 7), while LAGOM imparts a much smaller IVL-interval on the return value of Q, demonstrating that our lightweight synchronization preserves accuracy of results close to the sequential expectation.

B.5 Overall Takeaways

While IVL, as a useful correctness criterion, allows reasoning about semantics of concurrent queries and can preserve (ϵ, δ) bounds of sketches, it cannot alone fully characterize the accuracy of results. Our study shows that $\hat{F}_2{}^{\text{LAGOM}}$'s compensation scheme is very close to accurate estimations in a sequential setting, utilizing available memory for improved parallelism and accuracy in a concurrent one, where efficient synchronization permits low query latency, implying freshness of returned results. These observations illustrate the challenges tackled in this work.

Towards Optimal Distributed Edge Coloring with Fewer Colors

Manuel Jakob
TU Graz, Austria

Yannic Maus
TU Graz, Austria

Florian Schager
TU Graz, Austria

Abstract

There is a huge difference in techniques and runtimes of distributed algorithms for problems that can be solved by a sequential greedy algorithm and those that cannot. A prime example of this contrast appears in the edge coloring problem: while $(2\Delta - 1)$-edge coloring – where Δ is the maximum degree – can be solved in $\mathcal{O}(\log^*(n))$ rounds on constant-degree graphs, the seemingly minor reduction to $(2\Delta - 2)$ colors leads to an $\Omega(\log n)$ lower bound [Chang, He, Li, Pettie & Uitto, SODA'18]. Understanding this sharp divide between very local problems and inherently more global ones remains a central open question in distributed computing and it is a core focus of this paper.

As our main contribution we design a deterministic distributed $\mathcal{O}(\log n)$-round reduction from the $(2\Delta-2)$-edge coloring problem to the much easier $(2\Delta-1)$-edge coloring problem. This reduction is optimal, as the $(2\Delta - 2)$-edge coloring problem admits an $\Omega(\log n)$ lower bound that even holds on the class of constant-degree graphs, whereas the $2\Delta - 1$-edge coloring problem can be solved in $\mathcal{O}(\log^* n)$ rounds. By plugging in the $(2\Delta - 1)$-edge coloring algorithms from [Balliu, Brandt, Kuhn & Olivetti, PODC'22] running in $\mathcal{O}(\log^{12} \Delta + \log^* n)$ rounds, we obtain an optimal runtime of $\mathcal{O}(\log n)$ rounds as long as $\Delta = 2^{\mathcal{O}(\log^{1/12} n)}$. Previously, such an optimal algorithm was only known for the class of constant-degree graphs [Brandt, Maus, Narayanan, Schager & Uitto, SODA'25]. Furthermore, on general graphs our reduction improves the runtime from $\widetilde{\mathcal{O}}(\log^3 n)$ to $\widetilde{\mathcal{O}}(\log^{5/3} n)$.

In addition, we also obtain an optimal $\mathcal{O}(\log \log n)$-round randomized reduction of $(2\Delta - 2)$-edge coloring to $(2\Delta - 1)$-edge coloring. This leads to a $\widetilde{\mathcal{O}}(\log^{5/3} \log n)$-round $(2\Delta - 2)$-edge coloring algorithm, which beats the (very recent) previous state-of-the-art taking $\widetilde{\mathcal{O}}(\log^{8/3} \log n)$ rounds from [Bourreau, Brandt & Nolin, STOC'25].

Lastly, we obtain an $\mathcal{O}(\log_\Delta n)$-round reduction from the $(2\Delta - 1)$-edge coloring, albeit to the somewhat harder maximal independent set (MIS) problem.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed graph algorithms, edge coloring, LOCAL model

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.37

Related Version *Full Version*: https://arxiv.org/pdf/2504.13003 [51]

Funding This research was funded in whole or in part by the Austrian Science Fund (FWF) https://doi.org/10.55776/P36280, https://doi.org/10.55776/I6915.

Acknowledgements For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

1 Introduction

There is a huge difference in the techniques and runtimes of distributed algorithms for locally checkable problems that can be solved by a sequential greedy algorithm and those that cannot. For example, on bounded degree graphs, the former problems have distributed complexity



© Manuel Jakob, Yannic Maus, and Florian Schager;
licensed under Creative Commons License CC-BY 4.0



$\Theta(\log^* n)$ [24, 54, 60] while the latter require at least $\Omega(\log n)$ time [17, 19]. Understanding this sharp and potentially wide divide between very locally solvable problems and inherently more global ones remains a central open question in distributed computing.

A prime example of this contrast – explained in detail next and forming the main topic of this paper – appears in the edge coloring problem. A *k-edge coloring* of a graph $G = (V, E)$ is a function $\varphi : E \to \{1, \ldots, k\}$ such that no two adjacent edges receive the same color. This problem [60, 25, 28, 31, 4, 16], but also other *greedy problems* like $(\Delta + 1)$-vertex coloring, maximal independent set, and maximal matching [56, 1, 11, 6, 20, 2, 65, 43, 29, 37], has been extensively studied in the field of distributed computing, particularly in the LOCAL model.

In this model the communication network is abstracted as an undirected graph $G = (V, E)$. Each vertex hosts a processor that may communicate with its neighbors in synchronous rounds. In one round, each processor may send unbounded-size messages to all its neighbors in G and perform unbounded local computations. The goal is to minimize the number of synchronous rounds until every node outputs (a local part of) the solution. The related CONGEST model additionally imposes a maximum message size of $\mathcal{O}(\log n)$ bits per message.

In order to make progress in many parts of the graph in parallel, many distributed algorithms crucially rely on the property that color choices made by other vertices do not affect the solvability on the remaining graph. While this property is satisfied for $k = 2\Delta - 1$ (hence we call this the *greedy threshold*), for any $k < 2\Delta - 1$ there are partial k-edge colorings that cannot be extended to the entire graph [5]. On the other hand of the spectrum, it is known that every graph admits a $(\Delta + 1)$-edge coloring by a celebrated result of Vizing [69].

The intuition that coloring with fewer colors is significantly more challenging – particularly in the distributed setting – is evidenced by strong lower bounds: Any deterministic edge coloring algorithm using less than $2\Delta - 1$ colors requires at least $\Omega(\log_\Delta n)$ rounds, and even randomized algorithms face an $\Omega(\log_\Delta \log n)$-round lower bound [17]. On trees, where these lower bounds are established, there is even a deterministic algorithm that solves the greedy version of the problem in strongly sublogarithmic time, specifically in $\mathcal{O}(\log^{12/13} n)$ rounds [16]. On bounded degree graphs, the gap is even larger as the greedy version has deterministic and randomized complexity $\Theta(\log^* n)$, yet the lower bounds for algorithms that use fewer colors – $\Omega(\log n)$ deterministic and $\Omega(\log \log n)$ randomized – still apply.

Despite these challenges, there has been extensive work on designing increasingly faster deterministic and randomized algorithms for edge coloring with fewer than $2\Delta - 1$ colors – both in the LOCAL model [12, 13, 15, 17, 18, 44, 48], the more restricted CONGEST model [44, 47], and in other models such as streaming [5, 23, 66]. The fastest known deterministic distributed algorithm that operates below the greedy threshold runs in $\widetilde{\mathcal{O}}(\log^3 n)$ rounds[1] [15], while the fastest randomized algorithm, derived from a recent more general result on Δ-vertex coloring, achieves a runtime of $\widetilde{\mathcal{O}}(\log^{8/3} \log n)$ rounds [14]. See the related work section (Section 1.3) for further results on edge coloring with fewer colors.

This work aims to better understand the size and nature of the complexity gap between greedy-solvable problems and those below the greedy threshold. Understanding this latter class of problems is also central to the poorly understood regime in the complexity landscape of local graph problems, see, e.g., [30, 22, 58].

[1] We use the notation $\widetilde{\mathcal{O}}(f(n))$ to hide factors polylogarithmic in $f(n)$.

1.1 Our contributions

As our main contribution, we show that the considerably harder $(2\Delta-2)$-edge coloring problem can be efficiently reduced to its greedy cousin $(2\Delta-1)$-edge coloring[2] in $\mathcal{O}(\log n)$ rounds of the LOCAL model.

▶ **Theorem 1.** *There is a deterministic* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log n)+T_{2\Delta-1}(n,\Delta-1)$ *rounds.*

Secondly, allowing the use of maximal independent set (MIS) as a subroutine improves the runtime of our reduction at the cost of reducing to a harder problem.

▶ **Theorem 2.** *There is a deterministic* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log_\Delta n)+T_{\text{MIS}}(\Delta^2\cdot n,\text{poly}(\Delta))$ *rounds.*

While the latter result might sound like a strict improvement over the former – especially given the fact that the runtimes of the current state-of-the-art algorithms for $(2\Delta-1)$-edge coloring and MIS on general graphs coincide – we argue that the former version is, in fact, the stronger result. A classic reduction from Luby [56] shows that $(2\Delta-1)$-edge coloring is no harder than MIS. Consequently, the $\widetilde{\mathcal{O}}(\log^{5/3} n)$ deterministic algorithm for MIS [37] also solves the $(2\Delta-1)$-edge coloring problem. On the other hand, there is a $\mathcal{O}(\log^{12}\Delta+\log^* n)$-round $(2\Delta-1)$-edge coloring algorithm from [4], which provably cannot be achieved by any MIS algorithm due to the $\Omega(\min(\Delta,\log n/\log\log n))$ lower bound established in [3]. We obtain the following corollary.

▶ **Corollary 3.** *There are deterministic* LOCAL *algorithms computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *within either* $\mathcal{O}(\log^{12}\Delta+\log n)$ *or* $\widetilde{\mathcal{O}}(\log^{5/3} n)$ *rounds.*

For the class of graphs with maximum degree Δ as large as $2^{\mathcal{O}(\log^{1/12} n)}$, the first point of this corollary yields an optimal runtime of $\mathcal{O}(\log n)$ rounds. Previously, an optimal runtime was only known for the class of bounded-degree graphs [15]. The second point significantly improves the upper bound on the complexity of $(2\Delta-2)$-edge coloring on general graphs, which previously stood at $\widetilde{\mathcal{O}}(\log^3 n)$ rounds [15]. Additionally, we present an exponentially faster randomized reduction to the $(2\Delta-1)$-edge coloring problem.

▶ **Theorem 4.** *There is a randomized* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log\log n)+T^{\boxplus}_{2\Delta-1}(n,\Delta-1)$ *rounds.*

Theorem 4 yields the following corollary by plugging in the algorithms of [4] and [37].

▶ **Corollary 5.** *There are randomized distributed algorithms computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *within either* $\mathcal{O}(\log^{12}\Delta+\log\log n)$ *or* $\widetilde{\mathcal{O}}(\log^{5/3}\log n)$ *rounds of the* LOCAL *model.*

For the class of graphs with maximum degree Δ as large as $2^{\mathcal{O}(\log^{1/12}\log n)}$, the first point of this corollary yields an optimal runtime of $\mathcal{O}(\log\log n)$ rounds. The second point beats the (very recent) previous state-of-the-art for randomized $(2\Delta-2)$-edge coloring on general graphs, which previously stood at $\widetilde{\mathcal{O}}(\log^{8/3}\log n)$ rounds [14].

[2] We use $T_{2\Delta-1}(n,\Delta)$ ($T^{\boxplus}_{2\Delta-1}(n,\Delta)$) to denote the runtime of a deterministic (randomized) $(2\Delta-1)$-edge coloring algorithm and $T_{\text{MIS}}(n,\Delta)$ to denote the runtime of a deterministic MIS algorithm for n-vertex graphs with maximum degree Δ.

1.2 Our technique in a nutshell

To illustrate the difficulty of the $(2\Delta-2)$-edge coloring problem, let us first try to design an $\mathcal{O}(1)$-round reduction to the MIS and the $(2\Delta-1)$-edge coloring problem; recall that such a reduction provably cannot exist. One approach is to use the MIS algorithm to compute a clustering of the graph, e.g., by computing an MIS of the power graph G^2 and letting each vertex join the cluster of the closest node in the MIS; see Figure 1 for an example. As a result, we can partition the edge set of G into edges E_{intra} (black) within the clusters and inter-cluster edges E_{inter} (turquoise) that go between the different clusters. We could simply color the edges in E_{intra} independently and in parallel for each cluster, as the edges of different clusters are non-adjacent. Due to the constant diameter of each cluster, this can be done in $O(1)$ rounds and even using the existentially optimal number of colors, i.e., $\Delta+1$ colors. However, now the problem of coloring E_{intra} is no easier than the original $(2\Delta-2)$-edge coloring problem as it essentially corresponds to coloring a general graph but with some colors – the ones used by adjacent edges in E_{intra} – being forbidden for certain edges. In fact, it might not even be possible to complete the coloring at all.

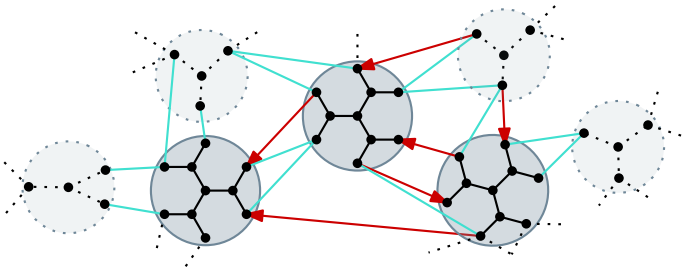


Figure 1 A matching of the intercluster edges that assigns two edges to each clusters. The matching edges are marked red. Arrows point towards the cluster the edge belongs to.

Instead, in our algorithm, we first color the edges between the clusters and then try to complete it within the clusters. First, observe that the graph induced by E_{inter} has maximum degree $\Delta' = \Delta - 1$, as each node has at least one adjacent edge within a cluster. Thus, we can color E_{inter} with one instance of greedy-type edge coloring, using only $2\Delta' - 1 = 2\Delta - 3$ colors. The main challenge that we address in this paper is how to ensure that the coloring can be extended to the edges within the clusters.

For the rest of the this technical overview we assume that the graph is Δ-regular with a sufficiently high girth such that each cluster looks like a tree; our full proof does not require this assumption. In addition, let us assume for now that each cluster is a perfect Δ-ary tree, that is, all leaves are at the same level. Then, a simplified version of our main technical contribution gives a useful condition under which we can complete the coloring.

▶ **Lemma 6** (Simplified version of Lemma 14)**.** *Consider a $(2\Delta-2)$-edge colored graph except for a perfect Δ-ary subtree $T \subseteq G$. If the number of distinct colors appearing on the incident edges of any leaf in T combined is at least Δ, then the coloring can be extended to T.*

For the remainder of this paper we will refer to the above condition as the *colorful condition*. To build some intuition for it, consider the simplest possible case, where the set of uncolored edges forms a star graph $S \subseteq G$. The only case, in which φ cannot be extended to S is if each edge in S has the exact same set of $\Delta-1$ colors appearing among its adjacent edges, which we illustrate in Figure 2. By setting up a bipartite graph with available colors

on one side and edges on the opposite side, an application of Hall's marriage theorem shows that this is indeed the only bad case. In the proof of Lemma 14, we generalize this condition from uncolored stars to uncolored trees.

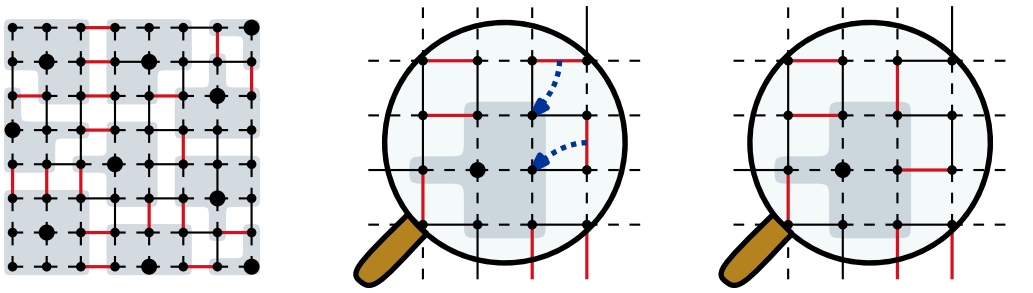


(a) This coloring can be extended. **(b)** This coloring cannot be extended.

Figure 2 Extendability of $(2\Delta - 2)$-colorings for $\Delta = 3$.

Next, we turn our attention to how the partial edge coloring of E_{inter} can be modified in order to satisfy the *colorful condition* for every cluster of uncolored edges. The core observation is that each leaf in the cluster already has $\Delta - 1$ incident edges in E_{inter}, whose colors must all be distinct. Therefore, if we can manage to keep the set of colors fixed for one leaf and modify at most one color at another leaf, the colorful condition is already satisfied. Notably, our partial coloring of E_{inter} only uses $2\Delta - 3$ of the $2\Delta - 2$ available colors. As a result, every edge already has at least one free color it can switch to if needed.

The main difficulty now is to construct a set of edges such that each cluster can perform these changes in parallel and without causing any conflicts. Hence, our desired set of edges needs to satisfy two key properties: (1) Each cluster needs exclusive access to two edges in its 1-hop neighborhood. (2) The set of all such edges needs to be a matching. Computing a greedy maximal matching satisfies (2), but not necessarily (1). Therefore, we modify a maximal matching $M \subseteq E_{\text{inter}}$ by letting each cluster send proposals to matching edges in its 2-hop neighborhood.



(a) The intracluster edges E_{intra} are dashed. We first compute a maximal matching $M \subseteq E_{\text{inter}}$, which is highlighted red.

(b) Next, we assign two edges in M to each cluster. This is the only "non-greedy" part of our reduction.

(c) Finally, we show how each cluster can move those edges into its neighborhood while maintaining the matching property.

Figure 3 In order to complete the coloring on the edges within the clusters, our algorithm ensures that each cluster can independently control the colors of two edges adjacent to the cluster. For illustration purposes this example has many cycles that are dealt with separately in our algorithm.

To solve this non-greedy problem of assigning matching edges to clusters, we set up a hypergraph H. Each cluster in G corresponds to a vertex in H and each edge $e \in M$ defines a hyperedge containing all clusters that sent at least one proposal to e. Note that a

hypergraph is in fact necessary here, since each edge in M may receive proposals from up to 2Δ different clusters. Now our assignment problem can then be viewed as a hypergraph orientation problem, where each hyperedge is said to be outgoing for exactly one of its vertices and ingoing for all others. In this setting, the problem has already been studied in the literature under the name *hypergraph sinkless orientation*. As long as each cluster sends out sufficiently more proposals than any edge in M receives, it can be solved in $\mathcal{O}(\log_\Delta n)$ rounds deterministically and $\mathcal{O}(\log_\Delta \log n)$ rounds randomized [15]. This is the only "non-greedy" part of our reduction and it dominates the runtime. Every other step runs in $\mathcal{O}(\log^* n)$ rounds for constant-degree graphs, which shows that this is an essential part of our reduction in light of the $\Omega(\log_\Delta n)$ lower bound for edge-coloring with less than $2\Delta - 1$ colors.

Now as a final technical remark, while each cluster may own two assigned edges, these may not be directly adjacent to the cluster and controlling their color would still be useless in terms of obtaining solvability of the cluster. To solve this issue we will show that cluster C can move its assigned edges into its 1-hop neighborhood along a shortest path to C, satisfying (1), while maintaining the matching property (2). Finally, the colorful condition is satisfied for every cluster and we can complete the $(2\Delta - 2)$-edge coloring for each cluster in parallel.

1.3 Related work

Distributed graph coloring is one of the most extensively studied problems in the field of distributed computing. Many foundational results are covered in the excellent book by Barenboim and Elkin [8]. In this section, we survey both, results for greedy and non-greedy edge coloring, highlighting a range of deterministic and randomized algorithms.

Greedy distributed edge coloring

Deterministic algorithms. In a seminal paper from 1987, Linial showed that any deterministic 3-coloring algorithm on the n-cycle takes at least $1/2 \cdot \log^* n$ rounds in the LOCAL model [54]. Naor later extended this lower bound to apply to randomized algorithms as well [59]. On the upper bound side, Linial also presented an $\mathcal{O}(\log^* n)$-round $\mathcal{O}(\Delta^2)$-coloring algorithm [54]. Combined with the classical one-color-per-round reduction this directly yields a $(2\Delta-1)$-edge coloring algorithm in $\mathcal{O}(\Delta^2 + \log^* n)$ rounds. With some clever tricks, subsequent works [6, 9, 10, 60, 53, 33, 57] have improved on this simple color reduction scheme, culminating in the current state-of-the-art for $(2\Delta - 1)$-edge coloring using $\mathcal{O}(\log^{12} \Delta + \log^* n)$ rounds [4]. Another line of research focuses on the complexity of graph coloring as a function of n, independent of Δ. Early works by Panconesi & Srinivasan [61, 63] show that using network decompositions, one can find a $(2\Delta - 1)$-edge coloring in $\exp(\mathcal{O}(\sqrt{\log n}))$ rounds. A long standing open question in the field asked for $(2\Delta-1)$-edge coloring algorithm in polylogarithmic time. Barenboim and Elkin made progress by showing that the $\Delta^{1+\varepsilon}$-coloring problem can be solved in $\mathcal{O}(\log^2 n)$ rounds [7]. Ghaffari & Su later gave a $\text{poly}(\log n)$ algorithm for $(2\Delta - 1)(1 + o(1))$-edge coloring [45], which in turn was improved by [42]. Fischer, Ghaffari, and Kuhn [31] fully resolved the question with a breakthrough $\mathcal{O}(\log^9 n)$-round algorithm based on a novel hypergraph maximal matching procedure. Since then, the field has picked up the pace with a lot of new algorithms being developed in the last decade, e.g. [4, 29, 36, 37, 39, 43, 48, 49]. The fastest algorithm for general graphs currently comes from the MIS algorithm in [37] and runs in $\widetilde{\mathcal{O}}(\log^{5/3} n)$ rounds.

Randomized algorithms. The first randomized algorithm for greedy edge coloring follows from the seminal $\mathcal{O}(\log n)$-round MIS algorithm independently developed by Luby [56] and Alon, Babai and Itai [1]. This upper bound stood for almost three decades until it was

surpassed by the $\mathcal{O}(\log \Delta) + 2^{\mathcal{O}(\sqrt{\log \log n})}$-round algorithm introduced in [11], pioneering the now ubiquitous shattering framework for randomized algorithms. This algorithm was later improved to $\mathcal{O}(\log^* \Delta) + T^{\text{det}}_{\text{deg}+1}(\text{poly} \log n)$ by Chang, Li, and Pettie [20]. Here, $T^{\text{det}}_{\text{deg}+1}(n')$ is the deterministic complexity of the $(\text{deg}+1)$-list coloring problem on instances of size n' [21]. A major breakthrough came with the first polylogarithmic network decomposition algorithm [65], reducing the complexity of the second term in both results and enabling $(2\Delta - 1)$-edge coloring in $\text{poly}(\log \log n)$ rounds. Subsequent works [29, 36, 37, 38, 64] continued to refine the runtime with the current state-of-the-art standing at $\widetilde{\mathcal{O}}(\log^{5/3} \log n)$ rounds [37].

Non-greedy distributed edge coloring

Despite the more challenging nature of the problem, there has also been a lot of progress in the non-greedy regime of coloring with $k < 2\Delta - 1$ colors. On the lower bound side, the authors of [17] established that coloring the edges of a graph with at most $2\Delta - 2$ colors takes at least $\Omega(\log_\Delta n)$ rounds deterministically and $\Omega(\log_\Delta \log n)$ rounds randomized. While most of the algorithms breaking the greedy color barrier were only developed in the last decade, there are two remarkable results before the turn of the millennium.

Δ-vertex coloring algorithms. In 1992, Panconesi & Srinivasan [61, 62] presented the first distributed Δ-coloring algorithm running in $\exp(\mathcal{O}(\sqrt{\log n}))$ rounds. This deterministic result also implies the first distributed $(2\Delta - 2)$-edge coloring algorithm. After two decades without improvements, the authors of [40] finally decreased the upper bound for the randomized complexity of Δ-coloring to $\mathcal{O}(\log \Delta) + 2^{\mathcal{O}(\sqrt{\log \log n})}$. Combining these seminal results with the new polylogarithmic algorithms for network decomposition [65, 38, 36, 29, 64, 37] yields runtimes of $\text{poly}(\log n)$ and $\mathcal{O}(\log \Delta) + \text{poly}(\log \log n)$, respectively. Recently, Bourreau, Brandt and Nolin [14] showed that Δ-coloring can be solved in $\mathcal{O}(\log^4 \Delta + \log^2 \Delta \cdot \log n \log^* n)$ deterministic time and $\widetilde{\mathcal{O}}(\log^{8/3} \log n)$ randomized time. The randomized version of this result is also the current state-of-the-art for $(2\Delta - 2)$-edge coloring. Moreover, for $\Delta = \omega(\log^{21} n)$ the randomized complexity of Δ-coloring collapses to $\mathcal{O}(\log^* n)$ due to [32]. Finally, a very recent result [50] gives an optimal Δ-coloring algorithm for a restricted graph class. However, since line graphs are not included in this class, it does not yield any new edge coloring results.

Randomized algorithms via the distributed Lovász Local Lemma. In 1998, Dubhashi, Grable & Panconesi [27] presented an $(1+\varepsilon)\Delta$-edge coloring algorithm using the *Rödl nibble method*, which runs in $\mathcal{O}(\log n)$ rounds, assuming $\Delta > (\log n)^{1+\gamma}$. After the turn of the millennium, there has been no progress on the problem, until Elkin, Pettie & Su [28] improved upon [27] in 2015. Their randomized $(1+\varepsilon)\Delta$-edge coloring algorithm reduces the problem to the distributed Lovász Local Lemma and runs in $\mathcal{O}(\log^* \Delta + \frac{\log n}{\Delta^{1-o(1)}})$ rounds. In both cases, ε need not be constant, but it is not clear how small it can be made as a function of Δ. Focusing on palette size over speed, the authors of [17] give a randomized $(\Delta + \widetilde{\Theta}(\sqrt{\Delta}))$-edge coloring algorithm that runs in $\text{poly}(\log n)$ rounds – nearly matching the natural threshold $\Delta + \Theta(\sqrt{\Delta})$, the smallest palette size for which there still is a constant probability of being able to color any edge given a random feasible coloring of its neighborhood. Recently, new advances in variants of the distributed Lovász Local Lemma [47, 26] have reduced the complexity of $(1+\varepsilon)\Delta$-edge coloring further to $\text{poly}(\log \log n)$.

Deterministic algorithms. In a 2018 paper [44], the authors present a $\mathcal{O}(\Delta^9 \cdot \text{poly}(\log n))$-time algorithm for $(3\Delta/2)$-edge coloring based on maximum matchings in bipartite graphs. Their result has since seen a series of improvements. First, Harris [49] presented improved

algorithms for hypergraph maximal matchings, reducing the runtime to $\widetilde{\mathcal{O}}(\Delta^4 \cdot \log^6 n)$ rounds. Next, a novel algorithm for hypergraph sinkless orientation [15] further reduced the complexity to $\mathcal{O}(\Delta^2 \cdot \log n)$ rounds. Using a splitting technique from [42, 41] this algorithm can also be adapted to a $(3/2+\varepsilon)\Delta$-edge coloring algorithm with runtime $\widetilde{\mathcal{O}}(\varepsilon^{-2} \cdot \log^2 \Delta \cdot \log n)$. Notably, this implies a $\mathcal{O}(\log^3 n)$-round algorithm for $(2\Delta-2)$-edge coloring in general graphs.

Vizing chain algorithms. A Vizing chain is an alternating path used to swap colors and resolve conflicts when trying to assign a new color to an edge in a graph that cannot be colored greedily. Vizing chains form a central role in Vizing's celebrated result stating that every graph is $(\Delta+1)$-edge colorable [69]. Recent years have brought significant progress on distributed versions of Vizing's theorem. In [68], the authors give the first $\text{poly}(\Delta, \log n)$ algorithms for randomized $(\Delta+2)$-edge coloring and deterministic $\Delta+\mathcal{O}(\log n/\log\log n)$ edge coloring. Further, Bernshteyn and Dhawan [12, 13] recently gave a deterministic $(\Delta+1)$-edge coloring algorithm in $\widetilde{\mathcal{O}}(\Delta^{84} \cdot \log^5 n)$ rounds, together with a faster randomized version running in $\mathcal{O}(\text{poly}(\Delta) \cdot \log^2 n)$ rounds. While the exponent in Δ in the runtime is certainly not yet optimized, there is a fundamental barrier to this approach. The authors of [17] have shown that it requires recoloring subgraphs of diameter $\Omega(\Delta \cdot \log n)$ in the worst case.

1.4 Organization of the paper

In Section 3 we present our main technical contribution: a sufficient condition under which partial $(2\Delta-2)$-edge colorings can be extended. Building on this, we introduce our new algorithms in Section 4. We start by presenting our reduction from $(2\Delta-2)$-edge coloring to MIS. Later on, in Section A, we replace the MIS and maximal matching subroutines to achieve a reduction to just $(2\Delta-1)$-edge coloring. Then, in Section B we adapt the reduction to obtain a faster randomized algorithm. For convenience, we collect all subroutines from the literature that we use in our algorithms in the appendix, to serve as a point of reference. The full version of our paper [51] also contains a table giving an overview of existing algorithms, the number of colors they use, and their runtime.

2 Preliminaries

We will use the notation $[k] := \{1, \dots, k\}$. Let $G = (V, E)$ be an undirected graph. The *neighborhood* of a vertex $v \in V$ in G is $N_G(v) = \{w \in V : vw \in E\}$ and the *edge-neighborhood* of v is $N_E(v) = \{e \in E : |e \cap \{v\}| = 1\}$. Further, for a subset of vertices $W \subseteq V$ we write $N_E(W) = \{e \in E : |e \cap W| = 1\}$. The *distance* between two vertices $u, v \in V$ is defined as the number of edges on a shortest path connecting u and v in G and is denoted by $\text{dist}(u, v)$. We define the distance between an edge $e = uv \in E$ and a vertex $w \in V$ as

$$\text{dist}(e, w) := \min(\text{dist}(u, w), \text{dist}(v, w)).$$

For any positive integer $k > 0$ we define the *power graph* $G^k = (V, E^k)$ by adding an edge for every pair of vertices $u, v \in V$ with $\text{dist}(u, v) \leq k$ to E^k. For any positive integer $k > 0$ and any vertex $v \in V$ we define the *k-hop neighborhood* of v to be $N^k(v) = N_{G^k}(v)$. For a subset of edges $F \subseteq E$ we write $G - F$ to denote the graph obtained by removing all edges in F from G. Further, for a subset $V' \subseteq V$, we denote by $G[V'] = (V', E')$ the subgraph of G *induced* by V'. The edge set E' is given by the subset of edges with both endpoints in V'. Similarly, for a subset of edges $E' \subseteq E$ we define the *edge-induced subgraph* $G[E'] = (V', E')$, where V' is given by the subset of vertices that are incident to at least one edge in E'. We

define the *degree* $\deg(e)$ of an edge $e \in E$ as the number of edges adjacent to e. A *hypergraph* is a generalization of a graph in which each edge can contain more than two vertices. The *rank* of a hyperedge e is the number of vertices contained in e. A *proper c-edge coloring* is a function $\varphi : E \to [c]$ that assigns different colors to adjacent edges. In this paper we will work a lot with *partial edge colorings* that assign colors only to a subset of edges.

▶ **Definition 7.** *Let $G = (V, E)$ be a graph, $F \subseteq E$ and $\varphi : E \setminus F \to [2\Delta - 2]$ a partial edge coloring of G. Then we call $\varphi^\star$ an* extension *of φ to F if $\varphi^\star$ is a proper $(2\Delta - 2)$-edge coloring of G and $\varphi^\star(e) = \varphi(e)$ for all $e \in E \setminus F$. Further, we say a color c is* available *for an edge $e \in F$, if no adjacent edge already has color c.*

▶ **Definition 8** ((α, β)-ruling set)**.** *For two integers $\alpha, \beta \geq 1$, an* (α, β)-ruling set *for a subset $W \subseteq V$ is a set of nodes $S \subseteq V$ such that the distance between any two nodes in S is at least α and any node $w \in W$ has a distance of at most β to the closest node in S. If $V = W$, we call S an (α, β)-ruling set.*

Ruling sets generalize maximal independent sets, which are $(2, 1)$-ruling sets. In our algorithm we will use ruling sets as a tool to partition the vertex sets into clusters with some specific properties. Most importantly, we will need that for each cluster C there is a central vertex $r \in C$ such that the whole k-hop neighborhood around r is contained in C, for some parameter $k > 0$. This ensures that each cluster has sufficiently many vertices, which will become important when we try to assign two exclusive matching edges to each cluster.

▶ **Definition 9** ((α, β)-clustering)**.** *Let $G = (V, E)$ be a graph and $\mathcal{R} \subseteq V$ a subset of vertices. Then, we call a partition $\bigcup_{r \in \mathcal{R}} C(r) = V$ of the vertex set into pairwise disjoint subsets a* (α, β)-clustering *with respect to $\mathcal{R}$ if $N^\alpha(r) \subseteq C(r)$ and $\operatorname{diam}(C(r)) \leq \beta$ for all $r \in \mathcal{R}$.*

▶ Remark 10. Computing a $(2, \beta)$-ruling set on the power graph G^k and letting each vertex join the cluster of one of its closest ruling set nodes leads to a $(\lfloor \frac{k}{2} \rfloor, k \cdot \beta)$-clustering.

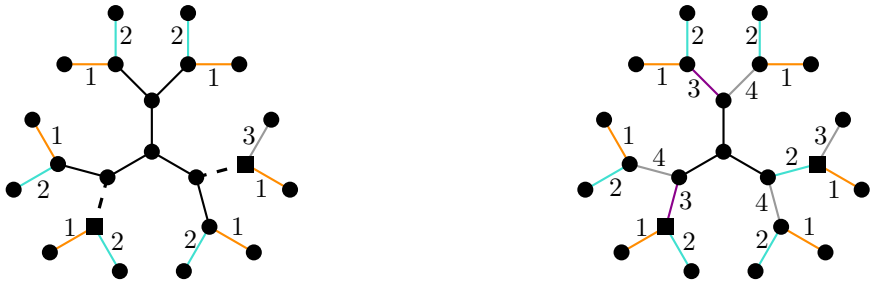
In the third step of our algorithm, our goal is to assign two exclusive edges to each cluster, such that the set of all such edges forms a matching in G. Starting with a maximal matching, we construct an auxiliary hypergraph H that lets us modify the matching in order to satisfy this property. This can be seen as an instance of the *hypergraph sinkless orientation* problem, which is a natural generalization of the fundamental *sinkless orientation* problem.

▶ **Definition 11** (Hypergraph sinkless orientation (HSO))**.** *Let $H = (V, E)$ be a hypergraph. The objective of the* hypergraph sinkless orientation *problem is to orient the hyperedges of H such that every vertex $v \in V$ has at least one outgoing hyperedge. We define an oriented hyperedge to be outgoing for exactly one of its incident vertices and incoming for all others.*

If the minimum degree $\delta = \min_{v \in V} \deg(v)$ and the maximum rank $r = \max_{e \in E} \operatorname{rank}(e)$ of H satisfy $\delta > r$, then this problem can be solved efficiently using the algorithms in Theorems 34 and 35. We follow the standard assumption that each vertex in the LOCAL model knows the number of nodes $n = |V|$ of the graph, as well as the maximum degree Δ.

3 Under which condition can we extend partial $(2\Delta - 2)$-colorings?

The goal of this section is to establish conditions under which we can extend φ to a proper $(2\Delta - 2)$-edge coloring of the entire graph G. As a first step, we examine the case where the subgraph induced by the uncolored edges is a simple star. In this setting, the only forbidden configuration occurs when all the leaves of the star graph share the same set of $\Delta - 1$ incident colors, which can be seen in Figure 2b.



(a) The edges with different colors palettes available are dashed.

(b) The coloring can be extended while using at least three colors for the newly colored edges.

Figure 4 Extending the coloring to a tree.

▶ **Lemma 12** (Reaching for the stars). *Let $G = (V, E)$ be a graph and let φ be a partial $(2\Delta - 2)$-edge coloring of G that only leaves the edges of a star graph $T = (V_T, E_T) \subseteq G$ uncolored. If $|\varphi(N_E(V_T))| \geq \Delta$, then there is an extension $\varphi^\star$ of φ to E_T.*

To prove this lemma we invoke the following classical combinatorial result.

▶ **Theorem 13** (Hall's theorem [46]). *Let $G = (V \dot\cup U, E)$ be a bipartite graph. For any subset $S \subseteq U$, let $N(S)$ denote the neighborhood of S in G. Then G has a U-saturating matching, if and only if $|N(S)| \geq |S|$ for all $S \subseteq U$.*

Proof of Lemma 12. We construct the auxiliary bipartite graph $B = (E_T \cup [2\Delta - 2], F)$ as follows: For every edge $e \in E_T$ and every color $c \in [2\Delta - 2]$ we add the edge (e, c) to F if and only if c is available for e under φ. Observe that any E_T-saturating matching in B corresponds to an extension of φ to E_T. By Theorem 13 such a matching exists if and only if for any subset $S \subseteq E_T$ it holds that $|N_B(S)| \geq |S|$. Since any edge in T has at least $\Delta - 1$ colors available and $|E_T| \leq \Delta$, the only set left to check is $S = E_T$. For contradiction sake, assume that $|N_B(E_T)| = \Delta - 1$ and let $r \in V_T$ be the center of the star. This implies that for every $e = rv \in E_T$ all colors in $C := [2\Delta - 2] \setminus N_B(E_T)$ are already occupied. Since $|C| = \Delta - 1 \geq |N_E(v) \cap N_E(V_T)|$, this is only possible if $\varphi(N_E(v) \cap N_E(V_T)) = C$ for all $e \in E_T$ and therefore $\varphi(N_E(V_T)) = C$. But $|\varphi(N_E(V_T))| \geq \Delta$, a contradiction! ◀

As the next step we show that this condition generalizes to trees as well. More precisely, we will show that if φ assigns at least Δ distinct colors to the edges adjacent to at least one vertex at distance k from the root, then there is always an extension of φ that assigns at least Δ distinct colors to the edges adjacent to any vertex at distance $k - 1$ from the root.

▶ **Lemma 14** (Colorful leaves make any tree happy). *Let $G = (V, E)$ be a Δ-regular graph, $T = (V_T, E_T) \subseteq G$ a tree, $r \in V_T$ and $\varphi : E \setminus E_T \to [2\Delta - 2]$ a partial edge coloring of G. Let*

$$V_k := \{v \in V_T : \mathrm{dist}_T(v, r) = k\} \qquad \text{and} \qquad E_k = \{e \in E_T : \mathrm{dist}_T(e, r) = k\}.$$

If there is a $k \in \mathbb{N}$ such that $|\varphi(N_E(V_k))| \geq \Delta$, then there exists an extension $\varphi^\star$ of φ to E_T.

Proof. Let $\ell \in \mathbb{N}$ such that $|\varphi(N_E(V_\ell))| \geq \Delta$. First we observe that for every $k > 0$, each edge $e \in E_k$ has exactly one neighboring edge in E_{k-1}. Therefore, we can always find at least one available color for every such edge, as long as the set E_{k-1} remains uncolored. Using this observation we can assign arbitrary available colors to all edges in $\bigcup_{k \geq \ell} E_k$ by processing these edge sets in sequence, starting with the set with highest index and going

through each of these sets in an arbitrary order. However, after we finish coloring E_ℓ we need to start being more deliberate with our color choices. For any $k < \ell$ we need to ensure that $|\varphi(N_E(V_k))| \geq \Delta$ holds after we colored all edges of E_k. This is due to the fact that we need $|\varphi(N_E(V_1))| \geq \Delta$ in order to apply Lemma 12, which guarantees that we can complete the coloring on the final layer of edges E_0 connected to the root. Now we show that given $|\varphi(N_E(V_{k+1}))| \geq \Delta$ we can color the edges in E_k in such a way that $|\varphi(N_E(V_k))| \geq \Delta$ for any $1 \leq k < \ell$. For each $e \in E_k$ let $N^+(e) := N_E(e) \cap N_E(V_{k+1})$. Since $|N^+(e)| = \Delta - 1$ we can always find two edges $e, e' \in E_k$ such that $\varphi(N^+(e)) \neq \varphi(N^+(e'))$. We first assign colors to all edges in E_k that are not adjacent to e or e'. Since we assume Δ to be sufficiently large, there always exists a vertex $v \in V_k$ that is not incident to either e or e'. Thus, after coloring all $\Delta - 1$ edges in $N_E(v) \setminus E_{k-1}$ incident at v we already have $|\varphi(N_E(V_k))| \geq \Delta - 1$. If $|\varphi(N_E(V_k))| \geq \Delta$, we are already done. Otherwise, if $|\varphi(N_E(V_k))| = \Delta - 1$, then the set $\varphi(N_E(V_k) \cup (N^+(e) \cap N^+(e')))$ contains at most $2\Delta - 3$ colors. Therefore, either e or e' must have a color available in $[2\Delta - 2] \setminus \varphi(N_E(V_k))$. We pick that color for e or e', which ensures that $|\varphi(N_E(V_k))| \geq \Delta$. The remaining edges in E_k can then be assigned arbitrary available colors again. For an illustration of this procedure we refer to Figure 4. Finally, to color the last layer E_0 containing the edges incident to the root, we can apply Lemma 12, since the number of distinct adjacent colors is at least Δ. ◀

The proofs of the next two lemmata argue in a similar way to previous work on Δ-vertex coloring and are therefore deferred to the appendix. First, we show that as soon as a cluster C contains a cycle of even length or a vertex of degree less than Δ, then we can extend any partial $(2\Delta - 2)$-edge coloring without modifications to $E(C)$.

▶ **Lemma 15.** *Let $G = (V, E)$ and $\varphi : E \to [2\Delta - 2] \cup \{\perp\}$ be a partial $(2\Delta - 2)$-edge coloring of G. Assume that the subgraph $H \subseteq G$ induced by the uncolored edges is connected and contains either a cycle of even length or a vertex of degree less than Δ. Then there exists an extension $\varphi^\star$ of φ, which is a proper $(2\Delta - 2)$-edge coloring of the entire graph G.*

On the other hand, if a cluster does not contain a cycle of even length, we need to ensure that it can satisfy the colorful condition from Lemma 14. To this end, it is crucial that the cluster contains a sufficient number of leaves. We apply a similar argument to the one used in the analysis of the Δ-vertex coloring algorithm in [40] to show that a connected component that does not contain an even cycle must be expanding exponentially.

▶ **Lemma 16.** *Let $G = (V, E)$ be a Δ-regular graph, $k > 0, v \in V$ and $C \subseteq G$ be a connected subgraph of G such that $N^k(v) \subseteq C$. Further assume that C does not contain an cycle of even length. Then, any BFS-tree T of C rooted at v contains at least $(\Delta - 2)^k$ leaves.*

4 Deterministic $(2\Delta - 2)$-edge coloring algorithm

We first present the simpler version of our algorithm, which reduces the $(2\Delta - 2)$-edge coloring problem deterministically to MIS in $\mathcal{O}(\log_\Delta n)$ rounds of the LOCAL model.

4.1 High level overview of our reduction to MIS

Recall that HSO (Theorem 34) aims to orient the hyperedges of a hypergraph such that each vertex has at least one outgoing hyperedge. In Phase 2 of Algorithm 1, it is used to rearrange the existing matching, ensuring that each cluster is assigned two exclusive edges.

Algorithm 1 Reduction from $(2\Delta-2)$-edge coloring to MIS (high level overview).

Input: a graph $G=(V,E)$
Output: $(2\Delta-2)$-edge coloring φ of G.

Phase 1: Partition vertices into clusters (Section 4.2)
1: Compute an MIS $\mathcal{I} \subseteq V$ on the power graph G^8 ▷ Theorem 36
2: $\mathcal{C}, E_T \leftarrow$ Cluster$(G, \mathcal{I})$ ▷ Algorithm 2
3: Compute a $(2\Delta-3)$-edge coloring φ of $G - E_T$ ▷ Lemma 18
Phase 2: Assign two exclusive edges to each cluster (Section 4.3)
4: Compute a maximal matching M of $G - E_T$ ▷ Lemma 19
5: Set up auxiliary hypergraph $H = (\mathcal{C}, M)$ ▷ Definition 21
6: Assign two matching edges to each cluster via HSO on H ▷ Theorem 34
7: Move the assigned edges into the neighborhood of the respective clusters ▷ Lemma 24
Phase 3: Switch colors in order to complete the coloring (Section 4.4)
8: **for** each cluster in parallel **do**
9: Change the colors of the assigned edges to fulfil the colorful condition ▷ Lemma 25
10: **end for**
11: Extend φ to a $(2\Delta-2)$-edge coloring $\varphi^\star$ of G ▷ Lemma 14

Phase 1: Partition vertices into clusters (Section 4.2). We start by computing a maximal independent set $\mathcal{I}$ on the power graph G^8. Next we partition the vertex set into disjoint clusters. Each vertex joins the cluster of one of its closest MIS-nodes, breaking ties arbitrarily, but consistently, which leads to a $(4,8)$-clustering $\mathcal{C}$. In addition, we store the BFS-tree (V, E_T) which arises naturally from this clustering. Computing the MIS on G^8 ensures that the individual clusters all contain sufficiently many vertices, which will become important later. Next, we remove the union of all BFS-trees from G to obtain a subgraph $G - E_T \subseteq G$ of maximum degree at most $\Delta - 1$. Hence, it suffices to invoke a greedy $(2\Delta-1)$-edge coloring on $G - E_T$ in order to get a $(2\Delta-3)$-edge coloring φ of $G - E_T$.

Phase 2: Assign two exclusive edges to each cluster (Section 4.3). Finally we need to modify φ such that the colorful condition from Lemma 14 is satisfied for every cluster. We will show that this can be achieved by giving each cluster control over the colors of two edges adjacent to the cluster. However, in order for the clusters to switch the colors of their respective assigned edges, the union of all those edge sets needs to be independent. Therefore we initially compute a maximal matching M of $G - E_T$ to use as a starting point. Note that the maximality of the matching does not guarantee that any fixed cluster receives any adjacent matching edges, but only that there are sufficiently many matching edges in the 2-hop neighborhood around that cluster. Hence, we then let each cluster propose to matching edges in its 2-hop neighborhood. This process naturally defines a hypergraph, where each matching edges collects the clusters which sent a proposal to it as vertices in a hyperedge. Then, the problem of assigning matching edges to clusters can be modeled as an instance of hypergraph sinkless orientation. In order to be able to solve HSO efficiently we need to ensure that the number of votes each matching edge receives is dominated by the number of votes each cluster casts in total. Here we require that each cluster contains enough vertices.

Phase 3: Switch colors in order to complete the coloring (Section 4.4). Finally, we show that each cluster can flip the colors of their assigned matching edges in parallel in order to fulfil the colorful condition from Lemma 14. Thus, this new coloring can now be extended to the remaining uncolored edges by iteratively coloring the layers of each BFS-tree in parallel. Since each cluster has diameter of at most 8 this last step can be executed in constant time.

▶ **Theorem 2.** *There is a deterministic* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log_\Delta n) + T_{\text{MIS}}(\Delta^2 \cdot n, \text{poly}(\Delta))$ *rounds.*

4.2 Phase 1: Partition vertices into clusters

Algorithm 2 CLUSTER$(G, \mathcal{I})$.

Input: a graph $G = (V, E)$ and an independent set $\mathcal{I} \subseteq V$
Output: a $(4, 8)$-clustering $\mathcal{C} = \{C^{-1}(r) : r \in \mathcal{I}\}$ and a forest $T = (V, E_T)$

$E_T \leftarrow \emptyset$
for each $v \in V$ in parallel **do**
  $C(v) \leftarrow \perp$
  if $v \in \mathcal{I}$ **then**
    $C(v) \leftarrow v$
  end if
  while $C(v) = \perp$ **do**
    receive messages from neighbors
    if $\exists\, u \in N(v) : C(u) \neq \perp$ **then**
      $C(v) \leftarrow C(u)$
      $E_T \leftarrow E_T \cup \{uv\}$ ▷ break ties arbitrarily
      send message $C(v)$ to neighbors
    end if
  end while
end for

▶ **Lemma 17** (Clustering algorithm)**.** *The set* $\mathcal{C} = \{C^{-1}(r) : r \in \mathcal{I}\}$ *computed by Algorithm 2 is a* $(4, 8)$*-clustering with respect to* $\mathcal{I}$*.*

Proof. First we show that $N^4(r) \subseteq C^{-1}(r)$ for all $r \in \mathcal{I}$. Let $r \in \mathcal{I}, v \in N^4(r)$ and assume for contradiction sake that $C(v) = r' \neq r$. Then, $\text{dist}(v, r') \leq \text{dist}(v, r) \leq 4$ and therefore $\text{dist}(r, r') \leq 8$, contradicting the fact that r and r' are independent in G^8. For the second property assume that there exist $r \in \mathcal{I}$ and $v \in C^{-1}(r)$ such that $\text{dist}(v, r) > 8$. This implies that $\text{dist}(v, r') \geq \text{dist}(v, r) > 8$ for all $r' \in \mathcal{I}$, because otherwise v would have joined the cluster of r' instead of r. But then v would be independent of all $r' \in \mathcal{I}$ on G^8, which contradicts the maximality of $\mathcal{I}$. ◀

For each $r \in \mathcal{I}$, let $T(r) = T[C^{-1}(r)]$ denote the BFS-tree of the cluster $C^{-1}(r)$ rooted at r.

▶ **Lemma 18.** *The subgraph* $G - E_T$ *has maximum degree of at most* $\Delta - 1$ *and can be* $(2\Delta - 3)$*-edge colored in* $T_{\text{MIS}}(\Delta^2 \cdot n, 4\Delta)$ *rounds of the* LOCAL *model.*

Proof. We observe that the edge set E_T returned by Algorithm 2 satisfies $E_T = \bigcup_{r \in \mathcal{I}} E(T(r))$. For every vertex $v \in V$ there is an $r \in \mathcal{R}$ such that $v \in C^{-1}(r)$. Since $T(r)$ is a spanning tree of $C^{-1}(r)$, we get that v is incident to at least one edge in $E(T(r))$. Thus, the maximum degree of $G - E_T$ is at most $\Delta - 1$. Using the standard reduction from vertex coloring to MIS [56] we can compute a $(2\Delta - 3)$-edge coloring of $G - E_T$ in $T_{\text{MIS}}(\Delta^2 \cdot n, 4\Delta)$ rounds. ◀

▶ **Lemma 19.** *We can compute a maximal matching* M *of the edges in* $G - E_T \subseteq G$ *deterministically in* $T_{\text{MIS}}(\Delta \cdot n, 2\Delta - 2)$ *rounds of the* LOCAL *model.*

Proof. A maximal matching of $G - E_T$ is simply an MIS on the line graph $L(G - E_T)$. ◀

4.3 Phase 2: Assign two exclusive edges to each cluster

In this phase, we rearrange the edges in the maximal matching M in order to assign two exclusive edges to each cluster, while preserving the matching property. These exclusive edges are essential for adjusting the coloring of the intercluster edges, as laid out in Lemma 25. This step relies on the HSO subroutine from Theorem 34, and we now detail the specific instance used in our algorithm. The high level idea is that each cluster sends proposals to nearby edges in the matching M. Each edge accepts exactly one proposal and we swap the edge towards the cluster that *won* it. This process can be modeled as an HSO instance where each matching edge in M represents an hyperedge that contains as vertices all clusters that sent a proposal to that edge. Since a directed hyperedge e is outgoing for exactly one of its vertices $v \in e$, we will say that v *won* the proposal for e.

However, the hypergraph sinkless orientation problem only guarantees at least one outgoing hyperedge for each vertex. Therefore, in order to achieve two exclusive hyperedges for each cluster, we simply split each cluster into two virtual nodes and let both participate in the HSO independently. In the following we formalize this idea and prove that this provides the desired properties as well as a deterministic runtime of $\mathcal{O}(\log n)$ rounds. First, we let each cluster $C^{-1}(r) \in \mathcal{C}$ that is not already edge-degree choosable select a subset of vertices $S \subseteq V(C^{-1}(r))$ to send proposals to a matching edges in their neighborhood. Let $\mathcal{C}'$ denote the set of clusters, which are not already edge-degree choosable. Crucially, in order to satisfy the colorful condition from Lemma 14 we require that the vertices in S are all leaves in the spanning tree $T(r) \subseteq C^{-1}(r)$ at the same distance from the root r.

▶ **Lemma 20.** *For each cluster $C^{-1}(r) \in \mathcal{C}'$, there exists an integer $k > 0$ and a subset $S \subseteq V(C^{-1}(r))$ of size at least $4 \cdot \Delta^3$ such that every vertex $v \in S$ is a leaf in the tree $T(r)$ that satisfies $\operatorname{dist}(r, v) = k$.*

Proof. Since $\mathcal{C}$ is a $(4, 8)$-clustering we have that $C^{-1}(r)$ necessarily contains all vertices in $N^4(r)$. Thus, according to Lemma 16 the corresponding BFS-tree contains at least $(\Delta - 2)^4$ leaves. Therefore, by the pigeonhole principle there exists a layer V_k for $k = 1, \ldots, 8$ such that V_k contains at least $(\Delta - 2)^4/8 > 4 \cdot \Delta^3$ leaves for Δ sufficiently large. ◀

For each cluster we select an arbitrary set S of at least $4 \cdot \Delta^3$ leaves which are all at the same distance from the center. The subset S is the subset of leaves that send requests. Each vertex in $v \in S$ sends exactly one proposal. If v is incident to a matching edge $e \in M$, then it proposes to e. Otherwise, v sends a proposal to an arbitrary matching edge that is one hop away from e. Such an edge must exist, due to the maximality of M.

▶ **Definition 21** (Auxiliary hypergraph)**.** *We define the* auxiliary hypergraph $H = (V_H, E_H)$*:*

- *For each cluster $C \in \mathcal{C}'$ we add a vertex v_C to V_H.*
- *For each matching edge $e \in M$ we add a hyperedge e_H to E_H that contains all vertices v_C for which there exists at least one vertex $v \in V(C)$ that proposed to e.*

▶ **Lemma 22.** *The minimum degree δ_H and the maximum rank r_H of H satisfy $\delta_H > \Delta \cdot r_H$.*

Proof. First we observe that a single matching edge $e = uv$ can get proposals only from u, v and vertices that are exactly one hop away from either u or v. Hence, there may be two proposals coming from u and v plus $2 \cdot (\Delta - 1)$ proposals coming from neighbors of u or v. Thus, the maximum rank of H is at most $r_H = 2\Delta$. Next, each cluster contains at least $4 \cdot \Delta^3$ vertices sending proposals. Since each vertex sends exactly one proposal and each edge receives at most 2Δ proposals the vertices of each cluster must propose to at least $2\Delta^2$ edges in total. Hence, the minimum degree of H can be lower bounded by $\delta = 2\Delta^2$. ◀

▶ **Lemma 23.** *We can compute an orientation of E_H such that each vertex in V_H has at least two outgoing hyperedges in $\mathcal{O}(\log_\Delta n)$ rounds deterministically and $\mathcal{O}(\log_\Delta \log n)$ rounds randomized.*

Proof. To compute an orientation with two outgoing edges per cluster we simply split each vertex into two virtual nodes with half the degree each. Thus, the minimum degree of the resulting hypergraph H' is still at least Δ^2. Therefore, we can solve HSO in $\mathcal{O}(\log_{\delta/r} n) = \mathcal{O}(\log_\Delta n)$ rounds deterministically and in $\mathcal{O}(\log_{\delta/r} \delta + \log_{\delta/r} \log n) = \mathcal{O}(\log_\Delta \log n)$ rounds randomized using the algorithms from Theorem 34 and Theorem 35. ◀

The orientation of H yields an assignment of matching edges to clusters such that each cluster receives exclusive access to two matching edges. However, the matching edges are not necessarily adjacent to their respective clusters yet. Therefore we modify the matching M by moving each edge closer to the vertex which won the proposal for it.

▶ **Lemma 24.** *There is a matching $M' = \chi(M)$ such that for all $e \in M$ it holds that $\chi(e)$ is adjacent to the vertex that won the proposal for e.*

Proof. For each $e \in M$, let v be the vertex (in the original graph G), which sent the winning proposal to e. If v is incident to e, then we simply set $\chi(e) = e$. Otherwise, let u be the endpoint of e which is adjacent to v and set $\chi(e) = uv$. With this procedure, we ensure that each cluster is adjacent to at least two edges in M' since each of the two partitions of the cluster receives at least one edge based on the HSO property. If a cluster is assigned more than two edges in M', it discards any edges beyond the required two. In order to show that the edge set M' is still a proper matching, we recall that every vertex $v \in S \subseteq V$ sends out exactly one proposal. Hence, there is no vertex that wins more than one edge in M. Assume that v won the proposal for the edge $e = uw$. If v is incident to e there is no change from M to M'. Otherwise, let u be adjacent to v. In this case, the edge $uw \in M$ is flipped to $vw \in M'$. Hence, v is the only vertex that gains an incident edge in M' compared to M. Since e is one of the closest edges to v in M there cannot be an edge $e' \in M$ that is already incident to v. Therefore every vertex that wins its only proposal is incident to exactly one edge in M'. On the other hand, every vertex that does not win its proposal cannot gain any additional adjacent edges in M'. Therefore M' is still a matching. ◀

4.4 Phase 3: Switch colors in order to complete the coloring

Now that each cluster has been assigned two exclusive edges located at the same distance from its central vertex, we show how to modify their colors to ensure that the colorful condition from Lemma 14 is satisfied for every cluster. To prove that this procedure can be applied to all clusters in parallel, we argue that the colorful condition can still be fulfilled even if the colors of all edges not adjacent to the two exclusive edges are modified adversarially.

▶ **Lemma 25** (Color switching). *Let $G = (V, E)$ be a Δ-regular graph, $T = (V_T, E_T) \subseteq G$ a tree, $r \in V_T$ and $\varphi : E \setminus E_T \to [2\Delta - 3]$ a partial edge coloring of G. Then, for any two independent edges $uv, u'v' \in E_k$, where u and u' are leaves in T, there exists a coloring $\varphi' : E \setminus E_T \to [2\Delta - 2]$ such that $|\varphi'(N_E(V_k))| \geq \Delta$ and $\varphi(e) = \varphi'(e)$ for all $e \notin \{uv, u'v'\}$.*

Proof. Recall that $V_k := \{v \in V_T : \mathrm{dist}_T(v, r) = k\}$. Since u and u' are leaves in T, they are incident to exactly $\Delta - 1$ edges in $E \setminus E_T$. Hence, $\varphi(N_E(V_k))$ already contains at least $\Delta - 1$ distinct colors. If $|\varphi(N_E(V_k))| \geq \Delta$ we simply set $\varphi' = \varphi$. Otherwise, if $|\varphi(N_E(V_k))| = \Delta - 1$, we set $\varphi'(uv) = 2\Delta - 2$. Since $\varphi(e) < 2\Delta - 2$ for all $e \in E_k$ we only need to argue that



(a) Before switching colors: $\varphi(N_E(V_1)) = \{1, 2\}$. **(b)** After switching colors: $\varphi(N_E(V_1)) = \{1, 2, 4\}$.

Figure 5 A simplified example for the color switching procedure, where $\Delta = 3$. The edges in the BFS-tree T_r are represented by solid lines, while edges outside of T_r are dashed. The assigned matching edges e_1, e_2 are marked orange.

the color $\varphi(uv)$ still remains in $\varphi'(E_k)$. Consider the sets $N := \{uw : uw \in E \setminus E_T\}$ and $N' := \{u'w : u'w \in E \setminus E_T\}$. Since u, u' are both leaves in T, both N and N' have cardinality $\Delta - 1$. Moreover, since $|\varphi(E_k)| = \Delta - 1$ and any two edges in N (or N') are adjacent, we have that $\varphi(N) = \varphi(N')$. Hence, there exists an edge $u'v''$ such that $\varphi(uv) = \varphi(u'v'') = \varphi'(u'v'') \in \varphi'(E_k)$. Thus, we have $\varphi'(E_k) = \Delta$. Note that this argument holds even if the colors of the edges not adjacent to u or v are modified adversarially. ◀

4.5 Proof of Theorem 2

▶ **Theorem 2.** *There is a deterministic* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log_\Delta n) + T_{\text{MIS}}(\Delta^2 \cdot n, \text{poly}(\Delta))$ *rounds.*

Proof. For $\Delta = \mathcal{O}(1)$ we apply the $(3\Delta/2)$-edge coloring algorithm from Theorem 39 to get a $(2\Delta-2)$-edge coloring in $\mathcal{O}(\log n)$ rounds. Hence, for the remainder of the proof we assume $\Delta \geq \Delta_0$ for a sufficiently large constant Δ_0. Since the power graph G^8 has n vertices and maximum degree polynomial in Δ, an MIS can be computed on it in time $T_{\text{MIS}}(n, \text{poly}(\Delta))$. Next, the clustering is possible in a constant number of rounds, since the radius of each cluster is at most 8. Also, both computing the $(2\Delta - 3)$-edge coloring φ of $G - E_T$ and finding a maximal matching in E_T can be reduced to MIS, and thus can be solved in time $T_{\text{MIS}}(\Delta^2 \cdot n, \text{poly}(\Delta))$. Then, according to Section 4.3 we can modify the matching in a way that each cluster receives two exclusive edges. The rearrangement can be modeled as an instance of HSO with minimum degree $\frac{\Delta^3}{4}$ and maximum rank Δ^2. Since each cluster has constant diameter, simulating this hypergraph incurs just a constant overhead factor. Thus, the deterministic HSO algorithm from Theorem 34 runs in $\mathcal{O}(\log_\Delta n)$ rounds. Next, we apply Lemma 25 to ensure that for each cluster there is a $k \leq 8$ such that $|\varphi(N_E(V_k))| \geq \Delta$. By Lemma 14 we can extend φ to a proper $(2\Delta - 2)$-edge coloring of G. ◀

References

1 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

2 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 481–497. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00037.

3 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *J. ACM*, 68(5):39:1–39:30, 2021. doi:10.1145/3461458.

4 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time polylogarithmic in Δ. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 15–25. ACM, 2022. doi:10.1145/3519270.3538440.

5 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Inf. Process. Lett.*, 44(5):251–253, 1992. doi:10.1016/0020-0190(92)90209-E.

6 Leonid Barenboim. Deterministic $(\Delta + 1)$-coloring in sublinear (in Δ) time in static, dynamic, and faulty networks. *J. ACM*, 63(5):47:1–47:22, 2016. doi:10.1145/2979675.

7 Leonid Barenboim and Michael Elkin. Distributed deterministic edge coloring using bounded neighborhood independence. *Distributed Comput.*, 26(5-6):273–287, 2013. doi:10.1007/S00446-012-0167-7.

8 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

9 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\Delta + 1)$-coloring and applications. *J. ACM*, 69(1):5:1–5:26, 2022. doi:10.1145/3486625.

10 Leonid Barenboim, Michael Elkin, and Fabian Kuhn. Distributed (delta+1)-coloring in linear (in delta) time. *SIAM J. Comput.*, 43(1):72–95, 2014. doi:10.1137/12088848X.

11 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.60.

12 Anton Bernshteyn. A fast distributed algorithm for $(\Delta + 1)$-edge-coloring. *J. Comb. Theory B*, 152:319–352, 2022. doi:10.1016/J.JCTB.2021.10.004.

13 Anton Bernshteyn and Abhishek Dhawan. Fast algorithms for vizing's theorem on bounded degree graphs. *CoRR*, abs/2303.05408, 2023. doi:10.48550/arXiv.2303.05408.

14 Yann Bourreau, Sebastian Brandt, and Alexandre Nolin. Faster distributed Δ-coloring via ruling subgraphs. *arXiv preprint arXiv:2503.04320*, 2025. doi:10.48550/arXiv.2503.04320.

15 Sebastian Brandt, Yannic Maus, Ananth Narayanan, Florian Schager, and Jara Uitto. On the locality of hall's theorem. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4198–4226. SIAM, 2025. doi:10.1137/1.9781611978322.143.

16 Sebastian Brandt and Ananth Narayanan. Towards optimal deterministic local algorithms on trees, 2025. doi:10.48550/arXiv.2505.01410.

17 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2633–2652. SIAM, 2018. doi:10.1137/1.9781611975031.168.

18 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed edge coloring and a special case of the constructive lovász local lemma. *ACM Trans. Algorithms*, 16(1), November 2019. doi:10.1145/3365004.

19 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM J. Comput.*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

20 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed $(\Delta+1)$-coloring algorithm? In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of*

the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018, pages 445–456. ACM, 2018. doi:10.1145/3188745.3188964.

21 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. Distributed $(\Delta + 1)$-coloring via ultrafast graph shattering. *SIAM Journal of Computing*, 49(3):497–539, 2020. doi:10.1137/19M1249527.

22 Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the LOCAL model. *SIAM J. Comput.*, 48(1):33–69, 2019. doi:10.1137/17M1157957.

23 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00010.

24 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control.*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

25 Andrzej Czygrinow, Michal Hanckowiak, and Michal Karonski. Distributed o(delta log(n))-edge-coloring algorithm. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 345–355. Springer, 2001. doi:10.1007/3-540-44676-1_29.

26 Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4273–4295. SIAM, 2023. doi:10.1137/1.9781611977554.CH163.

27 Devdatt P. Dubhashi, David A. Grable, and Alessandro Panconesi. Near-optimal, distributed edge colouring via the nibble method. *Theor. Comput. Sci.*, 203(2):225–251, 1998. doi:10.1016/S0304-3975(98)00022-X.

28 Michael Elkin, Seth Pettie, and Hsin-Hao Su. (2Δ - 1)-edge-coloring is much easier than maximal matching in the distributed setting. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 355–370. SIAM, 2015. doi:10.1137/1.9781611973730.26.

29 Salwa Faour, Mohsen Ghaffari, Christoph Grunau, Fabian Kuhn, and Václav Rozhon. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4409–4447. SIAM, 2023. doi:10.1137/1.9781611977554.CH168.

30 Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for lovász local lemma, and the complexity hierarchy. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.18.

31 Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.25.

32 Manuela Fischer, Magnús M. Halldórsson, and Yannic Maus. Fast distributed brooks' theorem. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2567–2588. SIAM, 2023. doi:10.1137/1.9781611977554.CH98.

33 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 625–634. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.73.

34 Beat Gfeller and Elias Vicari. A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs. In Indranil Gupta and Roger Wattenhofer,

editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 53–60. ACM, 2007. doi:10.1145/1281100.1281111.

35 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277. SIAM, 2016. doi:10.1137/1.9781611974331.CH20.

36 Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed MIS and approximate matching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1777–1790. ACM, 2023. doi:10.1145/3564246.3585243.

37 Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2148–2179. IEEE, 2024. doi:10.1109/FOCS61266.2024.00007.

38 Mohsen Ghaffari, Christoph Grunau, and Václav Rozhon. Improved deterministic network decomposition. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2904–2923. SIAM, 2021. doi:10.1137/1.9781611976465.173.

39 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 662–673. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00069.

40 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved distributed delta-coloring. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 427–436. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212764.

41 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Yannic Maus, Jukka Suomela, and Jara Uitto. Improved distributed degree splitting and edge coloring. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.19.

42 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Yannic Maus, Jukka Suomela, and Jara Uitto. Improved distributed degree splitting and edge coloring. *Distributed Comput.*, 33(3-4):293–310, 2020. doi:10.1007/S00446-018-00346-8.

43 Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1009–1020. IEEE, 2021. doi:10.1109/FOCS52979.2021.00101.

44 Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 418–430. ACM, 2018. doi:10.1145/3188745.3188906.

45 Mohsen Ghaffari and Hsin-Hao Su. Distributed degree splitting, edge coloring, and orientations. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2505–2523. SIAM, 2017. doi:10.1137/1.9781611974782.166.

46 Philip Hall. On representatives of subsets. *Classic Papers in Combinatorics*, pages 58–62, 1987.

47 Magnús M. Halldórsson, Yannic Maus, and Alexandre Nolin. Fast distributed vertex splitting with applications. In Christian Scheideler, editor, *36th International Symposium on Distributed*

Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA, volume 246 of *LIPIcs*, pages 26:1–26:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2022.26.

48 David G. Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 700–724. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00048.

49 David G. Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. *SIAM J. Comput.*, 49(4):711–746, 2020. doi:10.1137/19M1279241.

50 Manuel Jakob and Yannic Maus. Towards optimal distributed delta coloring. *CoRR*, abs/2504.03080, 2025. doi:10.48550/arXiv.2504.03080.

51 Manuel Jakob, Yannic Maus, and Florian Schager. Towards optimal distributed edge coloring with fewer colors, 2025. arXiv:2504.13003.

52 Fabian Kuhn, Yannic Maus, and Simon Weidner. Deterministic distributed ruling sets of line graphs. In Zvi Lotker and Boaz Patt-Shamir, editors, *Structural Information and Communication Complexity - 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, volume 11085 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2018. doi:10.1007/978-3-030-01325-7_19.

53 Fabian Kuhn and Roger Wattenhofer. On the complexity of distributed graph coloring. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 7–15. ACM, 2006. doi:10.1145/1146381.1146387.

54 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.20.

55 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

56 Michael Luby. A simple parallel algorithm for the maximal independent set problem. In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 1–10. ACM, 1985. doi:10.1145/22145.22146.

57 Yannic Maus and Tigran Tonoyan. Local conflict coloring revisited: Linial for lists. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.16.

58 Yannic Maus and Jara Uitto. Efficient CONGEST algorithms for the lovász local lemma. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 31:1–31:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.31.

59 Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM J. Discret. Math.*, 4(3):409–412, 1991. doi:10.1137/0404036.

60 Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Comput.*, 14(2):97–100, 2001. doi:10.1007/PL00008932.

61 Alessandro Panconesi and Aravind Srinivasan. Improved distributed algorithms for coloring and network decomposition problems. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 581–592. ACM, 1992. doi:10.1145/129712.129769.

62 Alessandro Panconesi and Aravind Srinivasan. The local nature of delta-coloring and its algorithmic applications. *Comb.*, 15(2):255–280, 1995. doi:10.1007/BF01200759.

63 Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *J. Algorithms*, 20(2):356–374, 1996. doi:10.1006/JAGM.1996.0017.

64 Václav Rozhon, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1114–1121. IEEE, 2022. doi:10.1109/FOCS54457.2022.00107.

65 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 350–363. ACM, 2020. doi:10.1145/3357713.3384298.

66 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.109.

67 Johannes Schneider, Michael Elkin, and Roger Wattenhofer. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theor. Comput. Sci.*, 509:40–50, 2013. doi:10.1016/J.TCS.2012.09.004.

68 Hsin-Hao Su and Hoa T. Vu. Towards the locality of vizing's theorem. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 355–364. ACM, 2019. doi:10.1145/3313276.3316393.

69 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

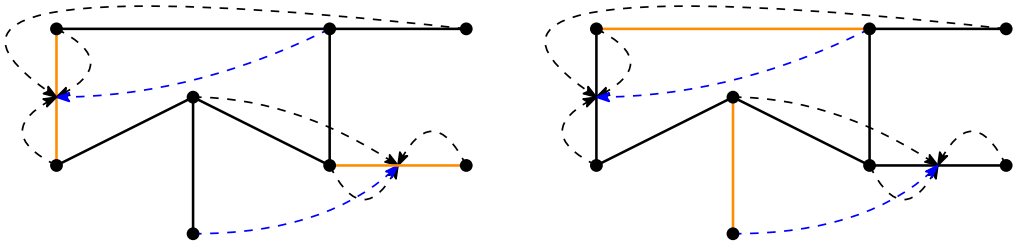
A Replacing MIS and maximal matching (Proof of Theorem 1)

In this subsection we show that computing an MIS is not strictly necessary for our reduction. Indeed, using a $(2, k)$-ruling set changes only the diameter of our clusters. Since increasing the cluster diameter increases the overhead for simulating the auxiliary hypergraph, this induces a tradeoff between k and the runtime of computing a $(2, k)$-ruling set. Setting $k = \mathcal{O}(\log \Delta)$ turns out to be the best choice for our purposes. Secondly, we show how to substitute the maximal matching by a 2-edge ruling set. This change is technically more involved, as it requires us to generalize the definition of our auxiliary hypergraph. The necessary changes to Algorithm 1 are also highlighted red in Algorithm 3.

Algorithm 3 Reduction from $(2\Delta-2)$-edge coloring to $(2\Delta-1)$-edge coloring (high level overview).

1: Compute a $(2, \mathcal{O}(\log \Delta))$-ruling set $\mathcal{R}$ on G^8 ▷ Corollary 31
2: $\mathcal{C}, E_T \leftarrow$ Cluster$(G, \mathcal{R})$ ▷ Algorithm 2
3: Compute a $(2\Delta - 3)$-edge coloring φ of $G - E_T$ ▷ Lemma 18
4: Compute a 2-edge ruling set $E_{\mathcal{R}}$ of T ▷ Theorem 33
5: Compute 2-HSO $\psi : \mathcal{C} \to E_{\mathcal{R}}$ in $H = (\mathcal{C}, E_{\mathcal{R}})$ ▷ Theorem 34
6: Rearrange the matching edges via $\chi : E_{\mathcal{R}} \to E'_{\mathcal{R}}$ ▷ Lemma 24
7: **for** each cluster $\mathfrak{C} \in \mathcal{C}$ in parallel **do**
8: Change the colors of the edges in $\chi(\psi(\mathfrak{C}))$ if necessary ▷ Lemma 25
9: **end for**
10: Extend φ to a $(2\Delta - 2)$-edge coloring $\varphi^\star$ of G ▷ Lemma 14

Secondly, we show that we can replace the maximal matching M by a cheaper 2-edge ruling set $E_{\mathcal{R}}$ of $G - E_T$. In this case, every leaf-vertex of a cluster sees at least one ruling set edge in its 3-hop neighborhood. Here we need to be a little bit more careful in how we set up our hypergraph. Firstly, each vertex may only send one proposal to one of its closest ruling set edges. Secondly, each vertex may only propose to a ruling set edge $e \in E_{\mathcal{R}}$, if all the vertices on a shortest path towards this have proposed to e as well. For an illustration we refer to Figure 6.



(a) Each vertex proposes to one of the nearest matching edges (orange) within its two-hop neighborhood. Note that vertices lying on the shortest path to a matching edge will propose to the same edge. These proposals are represented by dashed arrows. According to the HSO result, each orange matching edge accepts exactly one proposal, which is indicated by a blue arrow.

(b) Based on the HSO result (blue arrows), the matching is shifted toward the vertices that won the matching edge. The first edge along the shortest path to these vertices (orange) becomes the new matching edge.

Figure 6 Illustration of a simplified example showing how the HSO is modeled as a proposal instance for assigning clusters two exclusive edges.

For each cluster $C \in \mathcal{C}$ we select an arbitrary set $S(C) \subseteq V(C)$ of at least $4 \cdot \Delta^3$ leaves which are all at the same distance from the center.

▶ **Definition 26** (Proposal scheme). *Each cluster $C \in \mathcal{C}$ sends proposals to matching edges according to the following rules:*

- *Each vertex $v \in S(C)$ proposes to exactly one matching edge $e \in E_{\mathcal{R}}$.*
- *If v proposes to e, there may not be an edge $e' \in E_{\mathcal{R}}$ that is closer to v than e.*
- *If v proposes to e, every vertex on the shortest path from v to e must also propose to e.*

▶ **Definition 27.** *Let $e \in E_{\mathcal{R}}$ and $v \in V$ be the vertex which sent the winning proposal to e.*

- *If v is incident to e, then we simply add e to M'.*
- *If v is one hop away from e, let u be the endpoint of e which is adjacent to v and add the edge uv to M'.*
- *If v is two hops away from e, let u be the next vertex on a shortest path from v to e and add the edge uv to M'.*

Now we prove an analogous result to Lemma 24.

▶ **Lemma 28.** *The edge set M' is a proper matching in $G - E_T$ and every edge $e \in M'$ is adjacent to the vertex that won the proposal for e.*

Proof. Recall that every vertex $v \in V$ sends out at most one proposal. Hence, there is no vertex that wins more than one edge in $E_{\mathcal{R}}$. Assume that v won the proposal for the edge $e = uw \in E_{\mathcal{R}}$. If v is incident to e there is no change between $E_{\mathcal{R}}$ and M'. If v is one hop away from e, then v is the only vertex that gains an incident edge in M' compared to $E_{\mathcal{R}}$. Since e is one of the closest edges to v in $E_{\mathcal{R}}$ there cannot be an edge $e' \in E_{\mathcal{R}}$ that is already

incident to v. In the final case that v is two hops away from e, both v and the next vertex u on a shortest path from v to e gain an incident edge in $E_{\mathcal{R}}$. Note that every vertex is at most two hops away from its closest edge in $E_{\mathcal{R}}$, since $E_{\mathcal{R}}$ is a 2-edge ruling set. By our proposal scheme, u must also have proposed to v. Therefore u cannot have won its proposal and u cannot be incident to an edge in $E_{\mathcal{R}}$. Hence, both v and u are incident to only one edge in M'. Now assume that there is another vertex $v' = v$ that adds the edge $v'u$ to M'. This means that v' must have proposed to the same edge as u. But v already won the proposal for this edge, a contradiction! Therefore every vertex is incident to at most one edge in M'. ◀

▶ **Theorem 1.** *There is a deterministic* LOCAL *algorithm computing a $(2\Delta-2)$-edge coloring in any n-vertex graph with maximum degree Δ in $\mathcal{O}(\log n) + T_{2\Delta-1}(n, \Delta-1)$ rounds.*

Proof. The $(2, \mathcal{O}(\log \Delta))$-ruling set can be computed in $\mathcal{O}(\log \Delta + \log^* n)$ rounds according to Corollary 31. As an immediate consequence, the diameter of our clusters now increases to $\mathcal{O}(\log \Delta)$. This implies that simulating the hypergraph now incurs a $\mathcal{O}(\log \Delta)$ factor overhead. Therefore the complexity of computing a HSO increases to $\mathcal{O}(\log_\Delta n \cdot \log \Delta) = \mathcal{O}(\log n)$ rounds. The $(2\Delta-3)$-edge coloring of $G - E_T$ remains unchanged. Further, we can also relax the requirements for our matching M to 2-edge ruling set, which can be computed in $\mathcal{O}(\log^* n)$ rounds. Therefore the total runtime of our reduction works out to be

$$\mathcal{O}(\log \Delta + \log^* n) + \mathcal{O}(\log \Delta) + T_{2\Delta-1}(n) + \mathcal{O}(\log^* n) + \mathcal{O}(\log \Delta \cdot \log n) = \mathcal{O}(\log n) + T_{2\Delta-1}(n).$$ ◀

▶ **Corollary 3.** *There are deterministic* LOCAL *algorithms computing a $(2\Delta-2)$-edge coloring in any n-vertex graph with maximum degree Δ within either $\mathcal{O}(\log^{12} \Delta + \log n)$ or $\widetilde{\mathcal{O}}(\log^{5/3} n)$ rounds.*

Proof. For the first claim we plug in the $\mathcal{O}(\log^{12} \Delta + \log^* n)$-round algorithm from Theorem 38 into Theorem 1. For the latter we plug in the $\widetilde{\mathcal{O}}(\log^{5/3} n)$-round MIS algorithm from Theorem 36 into Theorem 2 instead. ◀

B Randomized $(2\Delta - 2)$-edge coloring algorithm

In order to achieve a $\mathcal{O}(\log \log n)$-round randomized reduction, we only need to replace the ruling set algorithm by a faster randomized variant in Corollary 32 and the deterministic HSO algorithm by its randomized counterpart in Theorem 35. The changes to the optimized deterministic version are also highlighted red in Algorithm 4.

Algorithm 4 Randomized $(2\Delta - 2)$-edge coloring (high level overview).

1:	Compute a $(2, \mathcal{O}(\log \Delta) \cap \mathcal{O}(\log \log n))$-ruling set $\mathcal{R}$ on G^8	▷ Corollary 32
2:	$\mathcal{C}, E_T \leftarrow$ Cluster$(G, \mathcal{R})$	▷ Algorithm 2
3:	Compute a $(2\Delta-3)$-edge coloring φ of $G - E_T$	▷ Lemma 18
4:	Compute a 2-edge ruling set $E_{\mathcal{R}}$ of T	▷ Theorem 33
5:	Compute 2-HSO $\psi : \mathcal{C} \to E_{\mathcal{R}}$ in $H = (\mathcal{C}, E_{\mathcal{R}})$	▷ Theorem 35
6:	Rearrange the matching edges via $\chi : E_{\mathcal{R}} \to E'_{\mathcal{R}}$	▷ Lemma 24
7:	**for** each cluster $\mathfrak{C} \in \mathcal{C}$ in parallel **do**	
8:	Change the colors of the edges in $\chi(\psi(\mathfrak{C}))$ if necessary	▷ Lemma 25
9:	**end for**	
10:	Extend φ to a $(2\Delta-2)$-edge coloring of G	▷ Lemma 14

▶ **Theorem 4.** *There is a randomized* LOCAL *algorithm computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *in* $\mathcal{O}(\log\log n)+T^{\boxplus}_{2\Delta-1}(n,\Delta-1)$ *rounds.*

Proof. The $(2,\mathcal{O}(\log\Delta)\cap\mathcal{O}(\log\log n))$-ruling set can be computed in $\mathcal{O}(\log\log n)$ rounds according to Corollary 32. Hence, each cluster has diameter at most $\mathcal{O}(\log\log n)$ and can be computed in $\mathcal{O}(\log\log n)$ rounds as well. For the 2-edge ruling set the deterministic algorithm from Theorem 33 running in $\mathcal{O}(\log^* n)$ rounds is still fast enough for our purposes here. According to Theorem 35 and Section 4.3 the hypergraph sinkless orientation can be computed in $\mathcal{O}(\log_\Delta\log n)$ rounds on the cluster graph, where we incur a $\mathcal{O}(\log\Delta)$-round simulation overhead. Finally, the extension to the edges inside each cluster can be computed in $\mathcal{O}(\log\log n)$ rounds. Hence, the total runtime works out to be

$$\mathcal{O}(\log\log n)+T^{\boxplus}_{2\Delta-1}(n)+\mathcal{O}(\log^* n)+\mathcal{O}(\log\Delta\cdot\log_\Delta\log n)=\mathcal{O}(\log\log n)+T^{\boxplus}_{2\Delta-1}(n).\text{◀}$$

▶ **Corollary 5.** *There are randomized distributed algorithms computing a* $(2\Delta-2)$*-edge coloring in any* n*-vertex graph with maximum degree* Δ *within either* $\mathcal{O}(\log^{12}\Delta+\log\log n)$ *or* $\widetilde{\mathcal{O}}(\log^{5/3}\log n)$ *rounds of the* LOCAL *model.*

Proof. To prove the first claim we plug in the deterministic $\mathcal{O}(\log^{12}\Delta+\log^* n)$-round algorithm from Theorem 38 into Theorem 4, while the second claim uses the randomized $\widetilde{\mathcal{O}}(\log^{5/3}\log n)$-round $(2\Delta-1)$-edge coloring algorithm from Corollary 37. ◀

Similar to the deterministic case we could also get a $\mathcal{O}(\log_\Delta\log n)$-round reduction to MIS via replacing the $(2,\mathcal{O}(\log\Delta)\cap\mathcal{O}(\log\log n))$-ruling set by an MIS. However, applying the shattering framework of [35] to the deterministic MIS algorithm from [37] still takes $\mathcal{O}(\log\Delta)+\widetilde{\mathcal{O}}(\log^{8/3}\log n)$ rounds. Hence, the additional cost of computing an MIS currently outweighs the time saved from the faster reduction.

C Deferred proofs

Proof of Lemma 15. Let C be a cycle of even length and assume that all edges in $N(C)$ have been colored already. Our task is now to find a proper edge-coloring of C, where every edge has a list of available colors of size two. If all edges have the same two colors available, then we can color the cycle alternately with these two colors. Otherwise, there exist two adjacent edges e_1,e_2 with different sets of available colors. Color e_1 with a color that is not available for e_2. Then we traverse the cycle starting at e_1 and going in the opposite direction of e_2 and let each edge pick one of its available colors. Since e_1 picked a color that is not available at e_2, there is also a color remaining for e_2 at the end. Since H is connected, every edge lies on a path towards an even cycle. Fix such an even cycle C and define

$$E_k=\{e\in E(H):\operatorname{dist}(e,C)=k\}.$$

Since every edge in E_k has at least one neighbor in E_{k-1} for $k>1$, we can greedily assign colors to these edges, starting from the highest layer. Finally, every edge in C has a list of two available colors and can therefore be properly colored as well. ◀

Proof of Lemma 16. For two nodes $u,u'\in V(C)$ let $P(u,u')$ denote the unique path in T between u and u'. We claim that each node $u\in V(C)$ is incident to at most one edge that is not part of $E(T)$ and thus the degree of all internal nodes of T is at least $\Delta-1$. First we observe that any edge in $E(C)\setminus E(T)$ can only connect vertices which are at most one level apart in T. Any additional edge in $E(C)\setminus E(T)$ from a vertex v_ℓ in layer ℓ to a

vertex $v_{\ell+1}$ in layer $\ell+1$ in T immediately induces an even cycle. To wit, let u be the least common ancestor of v_ℓ and $v_{\ell+1}$ and observe that the cycle $P(u,v_\ell) \cup \{v_\ell, v_{\ell+1}\} \cup P(u,v_{\ell+1})$ has length $2 \cdot |P(u,v_{\ell+1})|$. Next, let $u \in V(C)$ be a vertex in layer ℓ with two neighbors w, w' in the same layer ℓ. Then, $P(v,w) \cup \{w,u\} \cup \{u,w'\} \cup P(v,w')$ is a cycle of length $2(\ell+1)$. Hence each vertex $u \in V(C)$ is adjacent to at most one edge from $E(C) \setminus E(T)$. Since C contains the complete k-hop neighborhood of v this implies that there are at least $(\Delta-2)^k$ vertices in layer k of T. Every such vertex either has a leaf-descendant in T or is a leaf itself. ◀

D Complementary subroutines

All subroutines from the existing literature used throughout our algorithms are stated here explicitly.

D.1 Ruling sets

In this subsection we collect and combine various ruling set algorithms from the distributed literature that we will use as subroutines for our edge coloring algorithm.

▶ **Theorem 29** (ruling set algorithm [34, Theorem 8]). *There is a randomized distributed algorithm that computes a $(1, \mathcal{O}(\log\log\Delta))$-ruling set with induced degree at most $\mathcal{O}(\log^5 n)$ in any n-vertex graph with maximum degree Δ in $\mathcal{O}(\log\log\Delta)$ rounds*

▶ **Theorem 30** (ruling set algorithm [67, Theorem 3]). *Let $G=(V,E)$ and $W \subseteq V$. Given a d-coloring of G, there is a deterministic distributed algorithm that computes a $(2,c)$-ruling set for W in time $\mathcal{O}(c \cdot d^{1/c})$.*

▶ **Corollary 31.** *There is a deterministic distributed algorithm that computes a $(2, \mathcal{O}(\log\Delta))$-ruling set in any graph G in $\mathcal{O}(\log\Delta + \log^* n)$ rounds of the* LOCAL *model.*

Proof. We start by computing a $\mathcal{O}(\Delta^2)$-vertex coloring φ of G in $\mathcal{O}(\log^\star n)$ rounds using Linial's algorithm [54, 55]. Then, we apply Theorem 30 to φ for $c = \mathcal{O}(\log\Delta)$ to get a $(2, \mathcal{O}(\log\Delta))$-ruling set in time $\mathcal{O}(c \cdot d^{1/c}) = \mathcal{O}(\log\Delta \cdot \Delta^{2/\log\Delta}) = \mathcal{O}(\log\Delta)$. ◀

▶ **Corollary 32.** *There is a randomized distributed algorithm that computes a $(2, \mathcal{O}(\log\Delta) \cap \mathcal{O}(\log\log n))$-ruling set in any graph G in $\mathcal{O}(\log\log n)$ rounds of the* LOCAL *model.*

Proof. If $\Delta = \mathcal{O}(\log n)$, we can simply use the deterministic algorithm from Corollary 31 to get an $(2, \mathcal{O}(\log\Delta))$-ruling set (which is also a $(2, \mathcal{O}(\log\log n))$-ruling set) in $\mathcal{O}(\log\Delta + \log^* n) = \mathcal{O}(\log\log n)$ rounds. Otherwise, for $\Delta = \Omega(\log n)$ we first apply Theorem 29 to get a $(1, \mathcal{O}(\log\log\Delta))$-ruling set S_0 with induced degree at most $\mathcal{O}(\log^5 n)$. Next we compute a $\mathcal{O}(\log^{10} n)$-coloring χ of $G[S_0]$ using Linial's algorithm [54, 55]. Finally using χ as the input coloring and setting $c = \mathcal{O}(\log\log n)$, we get a $(2, \mathcal{O}(\log\log n))$ ruling set S_1 for S_0 in

$$\mathcal{O}(c \cdot d^{1/c}) = \mathcal{O}(\log\log n \cdot \log^{10/\log\log n} n) = \mathcal{O}(\log\log n).$$

randomized time. Since every vertex $v \in V$ is at most $\mathcal{O}(\log\log\Delta) = \mathcal{O}(\log\log n)$ hops away from a vertex in S_0 and every vertex in S_0 is at most $\mathcal{O}(\log\log n) = \mathcal{O}(\log\Delta)$ hops away from a vertex in S_1, we get that S_1 is both a $(2, \mathcal{O}(\log\Delta))$ and a $(2, \mathcal{O}(\log\log n))$-ruling set for V. ◀

Another subroutine that we will use in the optimized version of our algorithm computes a relaxation of a maximal matching.

▶ **Theorem 33** (2-ruling edge set algorithm [52, Theorem 1.3]). *There is a deterministic distributed algorithm that computes a 2-ruling edge set of G in $\Theta(\log^\star n)$ rounds in the* CONGEST *model.*

D.2 Hypergraph sinkless orientation

▶ **Theorem 34** (deterministic HSO [15, Theorem 1.4]). *There is a deterministic distributed algorithm for computing an HSO in any n-vertex multihypergraph $H = (V, E)$ with maximum rank r and minimum degree $\delta > r$ in $\mathcal{O}(\log_{\delta/r} n)$ rounds.*

▶ **Theorem 35** (randomized HSO [15, Theorem 1.5]). *There is a randomized distributed algorithm that w.h.p. computes an HSO on any hypergraph with maximum rank r and minimum degree $\delta \geq 320r \log r$ with runtime $\mathcal{O}(\log_{\delta/r} \delta + \log_{\delta/r} \log n)$.*

D.3 Maximal Independent Set

▶ **Theorem 36** (deterministic MIS [37, Theorem 1.3]). *There is a deterministic distributed algorithm that in any n-vertex graph $G = (V, E)$ computes an MIS in $\widetilde{\mathcal{O}}(\log^{5/3} n)$ rounds of the* LOCAL *model. This complexity also applies for maximal matching, $(\deg + 1)$-list vertex coloring, and $(2 \cdot \deg - 1)$-list edge coloring.*

▶ **Corollary 37** (randomized MIS [37, Theorem 1.3] & [20]). *There is a randomized distributed algorithm, in the* LOCAL *model, that computes a $(\Delta + 1)$-vertex coloring in $\widetilde{\mathcal{O}}(\log^{5/3} \log n)$ rounds. The same holds also for the $(2\Delta - 1)$-edge coloring problem.*

D.4 Edge coloring

▶ **Theorem 38** (deterministic $(2\Delta - 1)$-edge coloring [4, Theorem D.4]). *The $(\deg(e) + 1)$-list edge coloring problem can be solved in $\mathcal{O}(\log^7 C \cdot \log^5 n + \log^* n)$ deterministic rounds in the* LOCAL *model, where C is the size of the color space.*

▶ **Theorem 39** (deterministic $(3\Delta/2)$-edge coloring [15, Theorem 1.2]). *There is a deterministic $\mathcal{O}(\Delta^2 \cdot \log n)$-round* LOCAL *algorithm that computes a $(3\Delta/2)$-edge coloring on any n-vertex graph with maximum degree Δ.*

Compact Routing Schemes in Undirected and Directed Graphs

Avi Kadria
Bar Ilan University, Ramat Gan, Israel

Liam Roditty
Bar Ilan University, Ramat Gan, Israel

Abstract

In this paper, we study the problem of compact routing schemes in weighted undirected and directed graphs.

For weighted undirected graphs, more than a decade ago, Chechik [PODC'13] presented a $\approx 3.68k$-stretch compact routing scheme that uses $\tilde{O}(n^{1/k} \log D)$ local storage, where D is the normalized diameter, for every $k > 1$. We present a $\approx 2.64k$-stretch compact routing scheme that uses $\tilde{O}(n^{1/k})$ local storage *on average* in each vertex. This is the first compact routing scheme that uses total local storage of $\tilde{O}(n^{1+1/k})$ while achieving a $c \cdot k$ stretch, for a constant $c < 3$.

In real-world network protocols, messages are usually transmitted as part of a communication session between two parties. Therefore, more than two decades ago, Thorup and Zwick [SPAA'01] considered compact routing schemes that establish a communication session using a handshake. In their handshake-based compact routing scheme, the handshake is routed along a $(4k-5)$-stretch path, and the rest of the communication session is routed along an optimal $(2k-1)$-stretch path. It is straightforward to improve the $(4k-5)$-stretch of the handshake to $\approx 3.68k$-stretch using the compact routing scheme of Chechik [PODC'13]. We improve the handshake stretch to the optimal $(2k-1)$, by borrowing the concept of roundtrip routing from directed graphs to *undirected* graphs.

For weighted directed graphs, more than two decades ago, Roditty, Thorup, and Zwick [SODA'02 and TALG'08] presented a $(4k+\varepsilon)$-stretch compact roundtrip routing scheme that uses $\tilde{O}(n^{1/k})$ local storage for every $k \geq 3$. For $k = 3$, this gives a $(12+\varepsilon)$-roundtrip stretch using $\tilde{O}(n^{1/3})$ local storage. We improve the stretch by developing a 7-roundtrip stretch routing scheme with $\tilde{O}(n^{1/3})$ local storage. In addition, we consider graphs with bounded hop diameter and present an optimal $(2k-1)$-roundtrip stretch routing scheme that uses $\tilde{O}(D_{HOP} \cdot n^{1/k})$, where D_{HOP} is the hop diameter of the graph.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Routing schemes, Compact routing schemes, Distance oracles, Computer networks, Graph algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.38

Related Version *Full Version*: `https://arxiv.org/abs/2503.13753` [15]

Funding *Liam Roditty*: Supported in part by BSF grants 2016365 and 2020356.

1 Introduction

Routing is a fundamental task in computer networks. A *routing scheme* is a mechanism designed to deliver messages efficiently from a source vertex to a destination vertex within the network. In this paper, we study both undirected and directed weighted graphs, aiming to route along short paths.


© Avi Kadria and Liam Roditty;
licensed under Creative Commons License CC-BY 4.0



More specifically, a routing scheme is composed of a preprocessing phase and a routing phase. In the preprocessing phase, the entire graph is accessible, allowing the preprocessing algorithm to compute a routing table and a label for each vertex[1], which is then stored in the local storage of each vertex. In the routing phase, the routing algorithm at each vertex on the routing path can only access the local storage of the vertex. The routing algorithm gets as input a message, a destination label, and possibly a header, and decides which of the vertex neighbors is the next vertex on the routing path. The routing continues until the message reaches the destination.

A *compact* routing scheme is a routing scheme that uses $o(n)$ space in the local storage on average at each vertex, where n is the number of vertices in the graph. Let u be a source vertex and let v be a destination vertex. We denote by $\hat{d}(u,v)$ the length of the path used by the routing algorithm to route a message from u to v. The *stretch* of the routing scheme is defined as $\max_{u,v\in V}(\frac{\hat{d}(u,v)}{d(u,v)})$, where $d(u,v)$ is the distance from u to v. The *roundtrip* stretch is defined as $\max_{u,v\in V}(\frac{\hat{d}(u,v)+\hat{d}(v,u)}{d(u,v)+d(v,u)})$.

The design of efficient compact routing schemes in undirected graphs has been a well-studied subject in the last few decades, see for example [19, 3, 4, 8, 11, 24, 22, 7]. In Table 1 we summarize the previous results.

From the Erdős girth conjecture, it follows that every routing scheme with stretch $< 2k+1$ must use a total storage of $\Omega(n^{1+1/k})$ bits. The approximate distance oracle data structure of Thorup and Zwick [25], which is implemented in the centralized model, where all the information is available upon a distance query to the data structure, has an optimal $(2k-1)$-stretch with $\tilde{O}(n^{1+1/k})$ total storage. In light of the gap between routing schemes and approximate distance oracles, the following problem is natural.

▶ **Problem 1.** *For every $k \geq 2$, given a weighted undirected graph, what is the best stretch of a routing scheme that uses $\tilde{O}(n^{1+1/k})$ total storage?*

The $\approx 3.68k$-stretch compact routing scheme of Chechik [7], from more than a decade ago, is the current best stretch with $\tilde{O}(n^{1/k})$ worst-case local storage. In this paper, we improve the stretch to $\approx 2.64k$ by allowing an average local storage of $\tilde{O}(n^{1/k})$, as presented in the following theorem.

▶ **Theorem 1.** *Let $G=(V,E,w)$ be a weighted undirected graph. For every $k \geq 3$, there is an $\approx 2.64k$-stretch compact routing scheme that uses local routing tables of an average size of $\tilde{O}(n^{1/k})$, vertex labels of size $\tilde{O}(k)$ and packet headers of size $\tilde{O}(k)$.*

All the compact routing schemes mentioned so far solve the problem of sending a single message from the source to the destination, while in most real-world network applications, two parties communicate over the network for a session. A *communication session* is composed of two phases. In the first phase, a connection is established between the source and the destination, and in the second phase, a stream of messages is exchanged between the parties. Many real-world protocols, such as `TLS`, `QUIC`, `TCP`, `SSH`, `Wi-Fi`, and `Bluetooth`, adhere to this framework.

We consider the *handshake* mechanism for the establishment phase, as presented by Thorup and Zwick [24]. The handshake is composed of two messages of size $\tilde{O}(1)$, that are exchanged between the parties to establish the connection. The first message is sent from the

[1] In this paper, we study labeled routing schemes, where the preprocessing algorithm can assign labels to the vertices. When the vertex labels are fixed and cannot be changed, the routing scheme is called an name-independent routing scheme (see, for example, [1, 16, 17, 2, 13])

Table 1 Compact routing schemes in undirected graphs.

Stretch	Local storage	Uses average local storage?	Ref.	Comments
$O(k)$	$\tilde{O}(n^{1/k})$	yes	[19]	unweighted graphs
2^k-1	$\tilde{O}(n^{1/k})$	yes	[3]	
$O(k^2)$	$\tilde{O}(n^{1/k})$	no	[4]	
$4k-5$	$\tilde{O}(n^{1/k})$	no	[24]	
$\approx 3.68k$	$\tilde{O}(n^{1/k})$	no	[7]	
$4k-7+\varepsilon$	$\tilde{O}(n^{1/k})$	no	[22]	
$\approx 2.64k$	$\tilde{O}(n^{1/k})$	yes	Theorem 1	
$2k-1$	$\tilde{O}(n^{1/k})$	no	Theorem 2	roundtrip stretch

source to the destination, and the second message is sent from the destination back to the source. Since the handshake is composed of a message sent from the source to the destination and back, the stretch of the handshake is the roundtrip stretch defined earlier.

Thorup and Zwick [24] presented a compact routing scheme that uses a handshake, in which two messages are routed along a $(4k-5)$-stretch path, to establish a connection. Then, a stream of messages is routed along an optimal $(2k-1)$-stretch path. While the compact routing scheme of Chechik [7] achieves an $\approx 3.68k$-roundtrip stretch for the handshake, there is still a significant gap from the optimal $(2k-1)$-stretch followed by the Erdős girth conjecture. Therefore, the main open problem for routing in a communication session is to reduce the stretch of the handshake phase and obtain a compact roundtrip routing scheme with improved stretch.

▶ **Problem 2.** *For every $k \geq 2$, given a weighted undirected graph, what is the best roundtrip stretch of a routing scheme that uses $\tilde{O}(n^{1/k})$ local storage?*

In this paper we *solve* Problem 2 by presenting an *optimal* $(2k-1)$-roundtrip stretch for the handshake phase, that matches the lower bound that follows from the Erdős girth conjecture, as presented in the following theorem.

▶ **Theorem 2.** *Let $G=(V,E,w)$ be a weighted undirected graph. Let $k \geq 1$ be an integer. There is a $(2k-1)$-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(n^{1/k})$, vertex labels of size $\tilde{O}(k)$ and packet headers of size $\tilde{O}(k)$.*

Using this result with the handshake-based routing scheme of Thorup and Zwick [24], one obtains an optimal $(2k-1)$-stretch compact routing scheme for any communication session. We summarize our new results for undirected graphs and compare them to the previous work in Table 1.

Next, we turn our attention to weighted *directed* graphs. Since preserving the asymmetric reachability structure of directed graphs requires $\Omega(n^2)$ space[2], no spanner, emulator, or compact routing scheme can exist for directed graphs. Cowen and Wagner [9] circumvented the $\Omega(n^2)$ space lower bound by introducing roundtrip distances in directed graphs, defined as $d(u \leftrightarrow v) = d(u,v) + d(v,u)$.

[2] In a bipartite graph in which all the edges are from one side to the other, there are $\Theta(n^2)$ edges and removing each of them makes the destination not reachable from the source. Thus, storing reachability requires $\Omega(n^2)$ space.

In the last few decades, a few compact roundtrip routing schemes were presented, see for example [9, 10, 21]. The state-of-the-art result was obtained by Roditty, Thorup, and Zwick [21]. They presented a 3-stretch compact roundtrip routing scheme that uses $\tilde{O}(n^{1/2})$ local storage and also a $(4k+\varepsilon)$-stretch compact roundtrip routing scheme that uses $\tilde{O}(n^{1/k})$ local storage for every $k \geq 3$.

From the Erdős girth conjecture, it follows that every compact roundtrip routing scheme with stretch $< 2k+1$ must use total storage of $\Omega(n^{1+1/k})$ bits. Closing the gap between the upper and the lower bound is the main open problem regarding compact roundtrip routing schemes.

▶ **Problem 3.** *For every $k \geq 2$, given a directed weighted graph, what is the best stretch of a roundtrip routing scheme that uses $\tilde{O}(n^{1/k})$ local storage?*

In recent years, roundtrip distances have been extensively studied but only in the context of roundtrip spanners and roundtrip emulators (see, for example [14, 6, 21, 23, 5, 18]). Despite all the recent progress, no improvements were obtained for compact roundtrip routing schemes since the $(4k+\varepsilon)$-stretch roundtrip routing scheme of [21] from more than two decades ago.

In this paper, we improve upon [21] for the case that $k = 3$. More specifically, using $\tilde{O}(n^{1/3})$ local storage, Roditty, Thorup, and Zwick [21] obtained a $(12+\varepsilon)$-stretch roundtrip routing scheme. We present a 7-stretch roundtrip routing scheme that uses $\tilde{O}(n^{1/3})$ local storage, as presented in the following theorem.

▶ **Theorem 3.** *Let $G = (V, E, w)$ be a weighted directed graph. There is a 7-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(n^{1/3})$, vertex labels of size $\tilde{O}(1)$ and packet headers of size $\tilde{O}(1)$.*

In addition, in the following theorem, we present an optimal[3], up to polylogarithmic factors, compact roundtrip routing scheme in graphs with $D_{hop} = \tilde{O}(k)$, where D_{hop} is the hop diameter of the graph.

▶ **Theorem 4.** *Let $G = (V, E, w)$ be a weighted directed graph. Let $k \geq 1$ be an integer. There is a $(2k-1)$-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(D_{hop}n^{1/k})$, vertex labels of size $\tilde{O}(D_{hop}k)$ and packet headers of size $\tilde{O}(D_{hop})$.*

We summarize our new results for directed graphs and compare them to the previous work in Table 2.

The rest of this paper is organized as follows. In Section 2 we present some necessary preliminaries. In Section 3 we present an overview of our technical contributions and our new compact routing schemes. In Section 4 we present our optimal compact roundtrip routing scheme for weighted undirected graphs. In Section 5 we extend Section 4 to directed graphs with small hop diameter. In Appendix A we present our 7-stretch compact roundtrip routing scheme for weighted directed graphs. Finally, in Section 6 in the full version of this paper [15], we present our single message compact routing scheme for weighted undirected graphs that use average local storage.

[3] Assuming the Erdős conjecture, it is easy to create a graph G with $\Omega(n^{1+1/k})$ edges such that the girth of G is $2k+2$, and the diameter of G is at most $2k+1$. Given a graph with $\Omega(n^{1+1/k})$ edges and $2k+2$ girth, if there is a pair of vertices $u, v \in V$ such that $d(u,v) \geq 2k+2$, we can add an edge between u and v to the graph without reducing the girth. By repeating this process until there are no pairs $u, v \in V$ such that $d(u,v) \geq 2k+2$, we get that the diameter is at most $2k+1$.

Table 2 Compact roundtrip routing results in directed weighted graphs.[4]

Stretch	local storage	Uses average local storage?	Ref.	Comments
3	$\tilde{O}(n^{1/2})$	yes	[9]	
3	$\tilde{O}(n^{2/3})$	no	[9]	
2^k-1	$\tilde{O}(n^{1/k})$	yes	[10]	
3	$\tilde{O}(n^{1/2})$	no	[21]	
$4k+\varepsilon$	$\tilde{O}(\frac{1}{\varepsilon}n^{1/k})$	no	[21]	$\varepsilon > 0$
$12+\varepsilon$	$\tilde{O}(\frac{1}{\varepsilon}n^{1/3})$	no	[21]	$\varepsilon > 0$
7	$\tilde{O}(n^{1/3})$	no	Theorem 3	
$2k-1$	$\tilde{O}(D_{hop} \cdot n^{1/k})$	no	Theorem 4	D_{hop} is the hop diameter of G.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted graph with n vertices and m edges, where $w : E \to \mathbb{R}^+$. We consider both connected undirected graphs and strongly connected directed graphs[5].

Let the distance $d_G(u, v)$ from u to v be the length of the shortest path from u to v in G, where the length of a path is the sum of its edge weights, and let $P_G(u, v)$ be the shortest path from u to v, G is omitted when it is clear from context. The roundtrip distance $d(u \leftrightarrow v)$ is defined as $d(u, v) + d(v, u)$. Throughout this paper, we assume that for any two vertices u and v, there exists a unique shortest path between them. In the case of multiple shortest paths of the same length, we break ties by selecting the path with the lexicographically smallest sequence of vertex identifiers.

Let $X \subseteq V$. The distance $d(u, X)$ from u to X is the distance between u and the closest vertex to u from X, that is, $d(u, X) = \min_{x \in X}(d(u, x))$. Similarly, the roundtrip distance from u to X is defined as $d(u \leftrightarrow X) = \min_{x \in X}(d(u \leftrightarrow x))$. Let $p(u, X) = \arg\min_{x \in X}(d(u \leftrightarrow x))$ (ties are broken in favor of the vertex with a smaller identifier).

Next, following the ideas of Thorup and Zwick [25], we define bunches and clusters. Let $u \in V$ and let $X, Y \subseteq V$. The bunch of u with respect to X and Y is defined as $B(u, X, Y) = \{v \in X \mid d(u \leftrightarrow v) < d(u \leftrightarrow Y)\}$. The ball of u with respect to Y is defined as $B(u, Y) = \{v \in V \mid d(u \leftrightarrow v) < d(u \leftrightarrow Y)\}$ (notice that $B(u, Y) = B(u, V, Y)$). The cluster of u with respect to Y is defined as $C(u, Y) = \{v \in V \mid d(u \leftrightarrow v) < d(v \leftrightarrow Y)\}$.

The starting point in many algorithms, distance oracles, and compact routing schemes, and in particular in Thorup and Zwick [24] routing scheme, is a hierarchy of vertex sets $A_0, A_1, \ldots, A_k$, where $A_0 = V$, $A_k = \emptyset$, $A_{i+1} \subseteq A_i$ and $|A_i| = n^{1-i/k}$ for $0 \leq i \leq k-1$.

For every $0 \leq i \leq k-1$, the i-th pivot of u is defined as $p_i(u) = p(u, A_i)$, and $h_i(u)$ is defined as $d(u, A_i)$. The i-th bunch of u is defined as $B_i(u) = B(u, A_i, A_{i+1})$. The bunch of u is defined as the union of its individual bunches, that is, $B(u) = \bigcup_{i=0}^{k-1} B_i(u)$. The cluster of a vertex $w \in A_i \setminus A_{i+1}$ is defined as the cluster of w with respect to the set A_{i+1}, that is, $C(w) = C(w, A_{i+1})$. We denote by $[k]$ the set $\{0, 1, 2, \ldots, k-1\}$.

In the following lemma, we provide an upper bound for the size of $B(u)$, which is $O(kn^{1/k})$, as demonstrated by Thorup and Zwick [25].

[4] poly-logarithmic factors are omitted.

[5] If the graph is not connected (or not strongly connected), we add a dummy vertex and connect it with bi-directional edges of weight ∞ to every vertex of the graph.

▶ **Lemma 5** ([25, 20]). *Given an integer parameter $k \geq 2$, we can compute sets $A_1, \ldots, A_{k-1}$, such that $|A_i| = O(n^{1-i/k})$, for every $1 \leq i \leq k-1$. For every $i \in [k]$ the size of $B_i(u)$ is $\tilde{O}(n^{1/k})$.*

In the following lemma, we provide an upper bound for the size of $C(u)$, which is $O(n^{1/k})$, as demonstrated by Thorup and Zwick [24].

▶ **Lemma 6** ([24]). *Given a parameter p, we can compute a set A of size $\tilde{O}(np)$ such that, $|C(w, A)| = O(1/p)$, for every vertex $w \in V \setminus A$, and $|B(v, V, A)| = O(1/p)$ for every $v \in V$.*

Let $S \subseteq V$. We define $T_{out}(u, S)$ as a tree containing the directed shortest paths from u to all the vertices in S, and $T_{in}(u, S)$ as a tree containing the directed shortest paths from all the vertices in S to u. When u is clear from context, we omit it, for example, we use $T(C(u)) = T(u, C(u))$, and $T(B(u)) = T(u, B(u))$. Note that it is possible for $|T_{out}(u, S)|$ to be bigger than $|S|$ in cases where the shortest path from u to a vertex in S passes through a vertex not in S. [6] Let $T(u, X) = T_{in}(u, X) \cup T_{out}(u, X)$, as defined in [21], when u is clear from context we omit it and write $T(X)$. Notice that in undirected graphs, since $T_{in}(u, X) = T_{out}(u, X)$, we have that $T(u, X) = T_{in}(u, X) = T_{out}(u, X)$. Next, we show that if S is a ball, i.e. $S = B(u, X) = B(u, V, X)$ for some set X, then $|T_{out}(u, S)| = |S|$ and $|T_{in}(u, S)| = |S|$, and therefore $|T(u, S)| \leq 2|S|$.

▶ **Lemma 7** ([21]). *$|T_{out}(u, B(u, X))| = |B(u, X)|$ and $|T_{in}(u, B(u, X))| = |B(u, X)|$.*

Proof. Let $u \in V$, $v \in B(u, X)$, and let $w \in P(u, v)$. We will show that $w \in B(u, X)$. Since all the vertices in T_{out} are on the shortest path from u to a vertex in $B(u, X)$, we obtain that $|T_{out}(u, B(u, X))| \leq |B(u, X)|$, as required. The proof for the in-ball is identical for reversed paths. From the triangle inequality, we know that

$$d(u \leftrightarrow w) = d(u, w) + d(w, u) \overset{\triangle}{\leq} d(u, w) + d(w, v) + d(v, u) = d(u \leftrightarrow v) < d(u \leftrightarrow X),$$

where the last inequality follows from the fact that $v \in B(u, X)$.[7] Since $d(u \leftrightarrow w) < d(u \leftrightarrow X)$ we get that $w \in B(u, X)$, as required. ◀

The following lemma was originally proven in [25] for any metric space and therefore holds also for roundtrip distances. For completeness, we prove the lemma for roundtrip distances.

▶ **Lemma 8.** *Let $u, v \in V$ and let $0 < i \leq k-1$. If $p_j(u) \notin B(v)$ and $p_j(v) \notin B(u)$ for every $0 < j < i$, then*

$$d(u \leftrightarrow p_i(u)) \leq i \cdot d(u \leftrightarrow v) \quad \text{and} \quad d(v \leftrightarrow p_i(v)) \leq i \cdot d(u \leftrightarrow v).$$

Proof. We prove the claim by induction for every $0 \leq i \leq \ell$. For $i = 0$, the lemma holds since $d(v \leftrightarrow v) = 0$ and $d(u \leftrightarrow u) = 0$. Next, we prove the induction step. We assume the correctness of the claim for $i-1$ and show its correctness for i. Therefore, $d(v \leftrightarrow p_{i-1}(v)) \leq (i-1) \cdot d(u \leftrightarrow v)$ and $d(u \leftrightarrow p_{i-1}(u)) \leq (i-1) \cdot d(u \leftrightarrow v)$. Without loss of generality, we show that $d(u \leftrightarrow p_i(u)) \leq i \cdot d(u \leftrightarrow v)$. The proof for v is identical.

[6] We denote with $|H|$ the number of edges in H, i.e. $|H| = |E(H)|$.

[7] $\overset{\triangle}{\leq}$ denotes an inequality that follows from the triangle inequality.

Since $i \leq \ell$, it follows that $i-1 < \ell$. Therefore, from the lemma's assumptions, we know that $p_{i-1}(v) \notin B(u)$. From the definition of $B(u)$, it follows that $d(u \leftrightarrow p_i(u)) \leq d(u \leftrightarrow p_{i-1}(v))$. From the triangle inequality, it follows that $d(u \leftrightarrow p_{i-1}(v)) \leq d(u \leftrightarrow v) + d(v \leftrightarrow p_{i-1}(v))$. Recall that from the induction assumption we know that $d(v \leftrightarrow p_{i-1}(v)) \leq (i-1) \cdot d(u \leftrightarrow v)$. Therefore, we get that:

$$\begin{aligned} d(u \leftrightarrow p_i(u)) &\leq d(u \leftrightarrow p_{i-1}(v)) \leq d(u \leftrightarrow v) + d(v \leftrightarrow p_{i-1}(v)) \\ &\leq d(u \leftrightarrow v) + (i-1) \cdot d(u \leftrightarrow v) = i \cdot d(u \leftrightarrow v), \end{aligned}$$

as required. ◀

Next, we show that if $p_{i-1}(v) \notin B(u)$, then $d(v \leftrightarrow p_i(v)) \leq d(v \leftrightarrow p_{i-1}(v)) + 2 \cdot d(u \leftrightarrow v)$.

▶ **Lemma 9.** *Let $u, v \in V$, if $p_{i-1}(v) \notin B(u)$ then $d(v \leftrightarrow p_i(v)) \leq d(v \leftrightarrow p_{i-1}(v)) + 2d(u \leftrightarrow v)$*

Proof. From the definition of $p_i(v)$, we know that it is the closest vertex to v in A_i, and since $p_i(u) \in A_i$ we get that $d(v \leftrightarrow p_i(v)) \leq d(v \leftrightarrow p_i(u))$. From the triangle inequality, it follows that $d(v \leftrightarrow p_i(u)) \leq d(v \leftrightarrow u) + d(u \leftrightarrow p_i(u))$. From the lemma assumption, we know that $p_{i-1}(v) \notin B(u)$. Therefore, $d(u \leftrightarrow p_i(u)) \leq d(u \leftrightarrow p_{i-1}(v))$. From the triangle inequality, it follows that $d(u \leftrightarrow p_{i-1}(v)) \leq d(u \leftrightarrow v) + d(v \leftrightarrow p_{i-1}(v))$. Overall, we get that:

$$\begin{aligned} d(v \leftrightarrow p_i(v)) &\leq d(v \leftrightarrow p_i(u)) \leq d(v \leftrightarrow u) + d(u \leftrightarrow p_i(u)) \leq d(v \leftrightarrow u) + d(u \leftrightarrow p_{i-1}(v)) \\ &\leq d(v \leftrightarrow u) + d(u \leftrightarrow v) + d(v \leftrightarrow p_{i-1}(v)) = 2d(u \leftrightarrow v) + d(v, p_{i-1}(v)), \end{aligned}$$

as required. ◀

The following lemma holds only for *undirected* graphs and is presented in [25].

▶ **Lemma 10** ([25])**.** *Let $G = (V, E, w)$ be a weighted undirected graph. Let $v \in A_i \setminus A_{i+1}$, let $u \in C(v)$, and let $w \in P(v, u)$ then $w \in C(v)$.*

Proof. For the sake of contradiction, assume that $w \notin C(v)$. From the definition of $C(v)$, we have that $d(v \leftrightarrow w) \geq d(w \leftrightarrow A_{i+1})$.

By the triangle inequality, we have: $d(u \leftrightarrow A_{i+1}) \leq d(u \leftrightarrow w) + d(w \leftrightarrow A_{i+1})$. Since $d(w \leftrightarrow A_{i+1}) \leq d(v \leftrightarrow w)$, we get $d(u \leftrightarrow A_{i+1}) \leq d(u \leftrightarrow w) + d(w \leftrightarrow A_{i+1}) \leq d(u \leftrightarrow w) + d(v \leftrightarrow w)$. Since $w \in P(v, u)$, and the graph is undirected, it follows that $d(u \leftrightarrow w) + d(v \leftrightarrow w) = d(v \leftrightarrow u)$. Thus, $d(u \leftrightarrow A_{i+1}) \leq d(v \leftrightarrow u)$, a contradiction to the fact that $u \in C(v)$. ◀

2.1 General framework

A routing scheme comprises two phases: a preprocessing phase and a routing phase. In the preprocessing phase, the entire graph is accessible to the algorithm. The preprocessing algorithm computes for every $u \in V$ a routing table $\mathtt{RT}(u)$ and a label $\mathtt{L}(u)$. Each vertex u saves $\mathtt{RT}(u)$ and $\mathtt{L}(u)$ in its local storage. A routing scheme is considered *compact* if the size of the routing tables is sub-linear in the number of vertices, i.e., $|\mathtt{RT}(u)| = o(n)$.

In the routing phase, the goal is to route a message from a source vertex u to a destination vertex v. Specifically, the routing algorithm at the source vertex u gets as input the routing table $\mathtt{RT}(u)$, and the labels $\mathtt{L}(u)$ and $\mathtt{L}(v)$. Based on this input, the routing algorithm determines a neighboring vertex of u to which the message should be forwarded. The routing algorithm can also attach a header to the message. When a vertex w receives a message, the routing algorithm at w gets as input the routing table $\mathtt{RT}(w)$, and the labels $\mathtt{L}(w)$ and $\mathtt{L}(v)$ (and possibly a header). Based on this input, the routing algorithm determines a neighboring

vertex of w to which the message should be forwarded. The message is routed from a vertex to one of its neighbors until the message reaches its destination vertex v.

We denote by $\hat{d}(u,v)$ the distance that a message whose source vertex is u and whose destination vertex is v traverses from u to v. The stretch of the routing scheme is defined as $\max_{u,v\in V}(\frac{\hat{d}(u,v)}{d(u,v)})$.

Several variants of compact routing schemes exist. In a *labeled* routing scheme, the preprocessing algorithm can assign labels to the vertices. In a *fixed-port* routing scheme, the order of the neighbors of each vertex is fixed and cannot be changed by the preprocessing algorithm. In this work, we focus on labeled, fixed-port compact routing schemes.

2.2 Routing in trees

An essential ingredient in our compact routing schemes for general graphs is the following compact routing scheme for trees. Given a tree T, the preprocessing algorithm assigns a label $\mathtt{L}(v,T)$ to every vertex v in T. The routing algorithm then routes a message from a source vertex u to a destination vertex v on the shortest path from u to v in T. Thorup and Zwick [24] presented a tree routing scheme that uses only vertex labels of size $(1+o(1))\log n$ and no routing tables. In the fixed-port model, however, the label size increases to $O(\log^2 n)$. A similar scheme was presented by Fraigniaud and Gavoille [12]. The following lemma outlines the known results for tree compact routing schemes.

▶ **Lemma 11** ([12, 24]). *Let $T=(V,E)$ be an undirected tree on n vertices with each edge $e\in E$ assigned a unique $O(\log n)$-bit port number. Then, it is possible to efficiently assign each vertex $v\in V$ an $O(\log^2 n/\log\log n)$-bit label, denoted $label(v)$, such that if $u,v\in V$, then given $label(u)$ and $label(v)$, and nothing else, it is possible to find in constant time, the port number assigned to the first edge on the path in T from u to v.*

2.3 Routing in directed graphs

Roditty, Thorup, and Zwick [21] extended the compact routing scheme from trees to double trees to handle directed graphs. In our work on directed graphs (Section 5 and Appendix A), we employ this double-tree adaptation when routing within double-trees. Moreover, for the routing in clusters to work in directed graphs, they adjusted the definition of cluster adaptation using the following definition of roundtrip ordering.

▶ **Definition 12.** *We assume that $V=\{1,\ldots,n\}$. Let $u_1,u_2,v\in V$. We say that $u_1\prec_v u_2$ if one of the following holds:*

- $d(v\leftrightarrow u_1)<d(v\leftrightarrow u_1)$
- $d(v\leftrightarrow u_1)=d(v\leftrightarrow u_1)$ *and* $d(u_1\to v)<d(u_1\to v)$
- $d(v\leftrightarrow u_1)=d(v\leftrightarrow u_1)$ *and* $d(u_1\to v)=d(u_1\to v)$ *and* $u_1<u_2$

Using this definition, the cluster of u with respect to Y is defined as $C(u,Y)=\{v\in V\mid u\not\prec_v p_Y(u)\}$, where $p_Y(u)$ satisfies that $p_Y(u)\preceq_v w$ for every $w\in Y$.

Using this definition, they proved the following lemma:

▶ **Lemma 13** ([21]). *If $v\in C(u)$ and $w\in P(u,v)$, then $v\in C(w)$*

3 Overview

In this section, we present an overview of our new compact routing schemes and their main technical contributions. Throughout the overview, let $V=A_0,\ldots,A_k=\emptyset$ be a hierarchy such that, $|A_i|=\tilde{O}(n^{1-i/k})$ and $|B(u)|=\tilde{O}(n^{1/k})$, for every $u\in V$, created using Lemma 5. We denote with $x\to y$ routing from x to y on $P(x,y)$.

Roundtrip routing scheme in undirected graphs. Notice that any t-stretch compact routing scheme is also a t-roundtrip stretch compact routing scheme. In undirected graphs, where $d(u,v) = d(v,u)$, we have $d(u \leftrightarrow v) = d(u,v) + d(v,u) = 2d(u,v)$. This might lead one to question the potential benefits of considering roundtrip routing in undirected graphs. In other words, why might the problem of roundtrip routing be easier than the problem of single message routing, even though $d(u \leftrightarrow v) = 2d(u,v)$?

During the routing process, the information available to the routing algorithm at u may differ from the information available at v. Therefore, the routing path from u to v may differ from the routing path from v to u, which might lead to the case that $\hat{d}(u,v) \neq \hat{d}(v,u)$. Consider for example the case that $\hat{d}(u,v) = 3d(u,v)$ and $\hat{d}(v,u) = 27d(u,v)$. In this case, the roundtrip stretch is 15 while the single message stretch is 27, and even though $d(u \leftrightarrow v) = 2d(u,v)$, the roundtrip stretch is much smaller than the single message stretch.

Next, we provide an overview of our optimal $(2k-1)$-roundtrip stretch compact routing scheme (for the complete description, see Section 4). The preprocessing algorithm sets $\texttt{RT}(u) = \{\texttt{L}(u, T(C(v))) \mid v \in B(u)\}$ and $\texttt{L}(u) = \{\texttt{L}(u, T(C(p_i(u)))) \mid i \in [k]\}$, for every $u \in V$. Let $\ell(x,y) = \min\{i \mid p_i(y) \in B(x)\}$, and let $b = \ell(u,v)$, and let $a = \ell(v,u)$. The roundtrip routing path is $u \to p_b(v) \to v \to p_a(u) \to u$ (see Figure 1).

Our main technical contribution is in Lemma 16, where we show that while the path $u \to p_b(v) \to v$ might be of length at most $(4k-3)d(u,v)$, the entire path $u \to p_b(v) \to v \to p_a(u) \to u$ is of length of at most $(4k-2)d(u,v) = (2k-1)d(u \leftrightarrow v)$, and therefore the roundtrip stretch is the optimal $(2k-1)$-stretch.

Next, we provide some intuition why $\hat{d}(u \leftrightarrow v) \leq (2k-1)d(u \leftrightarrow v)$. From the triangle inequality, we have that $\hat{d}(u \leftrightarrow v) \leq d(u \leftrightarrow v) + d(u \leftrightarrow p_a(u)) + d(v \leftrightarrow p_b(v))$. From the definition of a and b, for every $0 < i < a \leq b \leq k-1$ we have $p_i(u) \notin B(v)$ and $p_i(v) \notin B(u)$, and for every $a \leq i < b$ we have $p_i(v) \notin B(u)$. This allows us to prove that $d(u \leftrightarrow p_a(u)) \leq a \cdot d(u \leftrightarrow v)$ and $d(v \leftrightarrow p_b(v)) \leq a \cdot d(u \leftrightarrow v) + (b-a) \cdot 2d(v \leftrightarrow p_b(v))$. Overall:

$$\begin{aligned} d(u \leftrightarrow p_a(u)) + d(v \leftrightarrow p_b(v)) &\leq ad(u \leftrightarrow v) + ad(u \leftrightarrow v) + (b-a)2d(u \leftrightarrow v) \\ &= 2bd(u \leftrightarrow v) \leq 2(k-1)d(u \leftrightarrow v) \end{aligned}$$

and therefore $\hat{d}(u \leftrightarrow v) \leq d(u \leftrightarrow v) + d(u \leftrightarrow p_a(u)) + d(v \leftrightarrow p_b(v)) \leq (2k-1)d(u \leftrightarrow v)$.

Directed roundtrip routing schemes. One might wonder why the general compact roundtrip routing scheme presented above for undirected graphs can not be extended to directed graphs. The problem lies in the structure of clusters. In undirected graphs, if $u \in C(w)$ then $P(u,w) \subseteq C(w)$ (see Lemma 10), and therefore we can route from u to every w, such that $u \in C(w)$. Unfortunately, in directed graphs, this nice property of clusters does not necessarily hold. More specifically, it might be that $P(u,w) \not\subseteq C(w)$ even when $u \in C(w)$ and as a result, we cannot route from u to w while using only the cluster $C(w)$ as in undirected graphs. A simple approach to overcome this problem is to store $P(u,w)$, for every $w \in B(u)$, in $\texttt{RT}(u)$. Using this approach, we can extend the roundtrip routing scheme from undirected graphs to directed graphs with small hop diameter (see Theorem 4).

For routing tables of size $\tilde{O}(n^{1/3})$, we overcome the problem that $P(u,w) \not\subseteq C(w)$ using a more sophisticated solution. For an hierarchy with $k = 3$ we have $V = A_0, A_1, A_2, A_3 = \emptyset$ and three bunches, $B_0(u), B_1(u)$ and $B_2(u)$, for every $u \in V$. Let $u, v \in V$. To follow the compact roundtrip of undirected graphs we need to be able to route from u to v, if $v \in B_0(u)$, from u to $p_1(v)$, if $p_1(v) \in B_1(u)$, and from u to $p_2(v)$, otherwise. Since $B_0(u)$ is a ball

$P(u,v) \subseteq B_0(u)$, for every $v \in B_0(u)$ (see Lemma 7). Therefore, we can route from u to every vertex in $B_0(u)$. Moreover, since $C(p_2(v)) = V$, we can route from u to $p_2(v)$. This leaves us with the challenge of routing from u to $p_1(v)$ when $p_1(v) \in B_1(u)$.

To handle this challenge, we divide the routing from u to $p_1(v)$ into two cases. If $P(u,p_1(v)) \subseteq C(p_1(v))$, we simply route on the path $P(u,p_1(v))$. Otherwise, if $P(u,p_1(v)) \nsubseteq C(p_1(v))$, we let $z \in P(u,p_1(v))$ be the first vertex such that $z \notin C(p_1(v))$. In this case, we route along the path $u \to p_2(z) \to p_1(v)$. Using the fact that $z \notin C(p_1(v))$ we show that $d(u,p_2(z)) + d(p_2(z),p_1(v)) \leq d(u,p_1(v)) + d(u \leftrightarrow p_1(v))$, and that $\hat{d}(u \leftrightarrow v) \leq 7d(u \leftrightarrow v)$ (see Lemma 19).

Average single message routing schemes in undirected graphs. Recall that $h_i(u) = d(u,A_i)$. In the preprocessing algorithm, we set $\texttt{RT}(u) = \{\texttt{L}(u,T(C(v))) \mid v \in B(u)\} \cup \{L(v,T(C(u))) \mid v \in C(u)\}$, and $\texttt{L}(u) = \{\langle h_i(u), \texttt{L}(u,T(C(p_i(u))))\rangle \mid i \in [k]\}$, for every $u \in V$.

For the sake of simplicity, we assume that when routing from the source vertex u to the destination vertex v, the value $d(u,v)$ is known to the routing algorithm. In this case, we present an optimal $(2k-1)$-stretch routing scheme in Section 6 of the full version of this paper [15]. The algorithm at the source u works as follows. For each $0 \leq i \leq k-1$, if $p_i(v) \in B_i(u)$, it routes from u to v in $T(C(p_i(v)))$. Otherwise, if the inequality $h_{i+1}(v) > d(u,v) + h_i(u)$ holds, then we have that $v \in C(p_i(u))$ and therefore the algorithm routes from u to v in $T(C(p_i(u)))$. If neither condition is satisfied, the algorithm proceeds to the next iteration. The $(2k-1)$-stretch of the routing scheme follows by induction using standard tools (see the full version[15] for the complete proof).

In Section 6 of the full version[15], we present a routing algorithm that routes from u to v without knowing the value of $d(u,v)$. To achieve this, we introduce an estimate $\hat{\delta}$ satisfying $\hat{\delta} \leq d(u,v)$, which serves as our current best lower bound for $d(u,v)$. Note that since $\hat{\delta}$ is only an estimate and may be strictly smaller than $d(u,v)$, it is possible that $h_{2i+1}(v) > \hat{\delta} + h_{2i}(u)$ and $v \notin C(p_{2i}(u))$.[8] Therefore, we introduce a condition that, if satisfied, means that $\hat{\delta} \ll h_{2i+1}(v) - h_{2i}(u)$. If the condition is satisfied, the routing algorithm routes to $p_{2i}(u)$ and checks in $\texttt{RT}(p_{2i}(u))$ whether $v \in C(p_{2i}(u))$. If $v \in C(p_{2i}(u))$ the algorithm simply routes from $p_{2i}(u)$ to v. Otherwise, if $v \notin C(p_{2i}(u))$, then we know that $h_{2i+1}(v) - h_{2i}(u) \leq d(u,v)$, and we can safely update $\hat{\delta}$ to $h_{2i+1}(v) - h_{2i}(u)$. We proceed by routing back from $p_{2i}(u)$ to u and then continuing to the next iteration of the algorithm.

The routing algorithm is simple and works as follows. Let $\ell = \min\{i \mid p_i(v) \in B(u)\}$, and let $\hat{\delta} = \max_{0 \leq i \leq \ell}(h_{i+1}(u) - h_i(v))$. For each $0 \leq i \leq \ell/2$, let $1 < c_i < 2$ be a constant:

1. If $h_{2i+1}(v) \leq c_i \cdot \hat{\delta} + h_{2i}(u)$, then the algorithm continues to the next iteration.
2. Otherwise, if $h_{2i+1}(v) > c_i \cdot \hat{\delta} + h_{2i}(u)$, then the algorithm routes to $p_{2i}(u)$ and accesses $\texttt{RT}(p_{2i}(u))$:
   a. If $v \in C(p_{2i}(u))$ then the algorithm routes directly from $p_{2i}(u)$ to v in $T(C(p_{2i}(u)))$.
   b. Otherwise, if $v \notin C(p_{2i}(u))$, the algorithm sets $\hat{\delta}$ to $h_{2i+1}(v) - h_{2i}(u)$ and routes back to u from $p_{2i}(u)$. The algorithm then continues to the next iteration.

In contrast to the simplicity of the routing algorithm, analyzing its stretch is rather involved. For a complete proof, see Section 6 of the full version[15].

[8] We alternate between bounding $h_i(u)$ and $h_i(v)$, but by the definition of $\hat{\delta}$, we have that $h_{i+1}(u) \leq \hat{\delta} + h_i(v)$ for every $i \leq \ell$. Therefore, it suffices to only bound the $h_{2i+1}(v) - h_{2i}(u)$ for $0 \leq i \leq a$.

4 Optimal roundtrip routing in undirected graphs

In this section, we consider roundtrip routing in weighted undirected graphs. In undirected graphs $d(u,v) = d(v,u)$, the roundtrip distance $d(u \leftrightarrow v)$ is simply $d(u,v)+d(v,u) = 2d(u,v)$.

This observation naturally leads to the question: what are the potential advantages of studying *roundtrip* routing in undirected graphs? In particular, why might roundtrip routing be easier to approximate than single-message routing, despite the fact that the roundtrip distance is always twice the one-way distance, i.e., $d(u \leftrightarrow v) = 2d(u,v)$?

The key distinction lies in the asymmetry of available information during the routing process. When routing from a source vertex u to a destination vertex v, the algorithm has access only to the routing table $\texttt{RT}(u)$ and the label $\texttt{L}(v)$. However, routing from v to u is based solely on $\texttt{RT}(v)$ and $\texttt{L}(u)$. Since these inputs may differ significantly, the resulting routing paths may differ significantly, and we may have that $\hat{d}(u,v) \neq \hat{d}(v,u)$.

In single-message routing, the stretch must hold for the worst-case direction. Therefore, $\frac{\max\left(\hat{d}(u,v),\hat{d}(v,u)\right)}{d(u,v)}$ is bounded. However, roundtrip routing requires only that the average of the two directions be bounded. Therefore, $\frac{\hat{d}(u\leftrightarrow v)}{d(u\leftrightarrow v)} = \frac{\hat{d}(u,v)+\hat{d}(v,u)}{2d(u,v)}$ is bounded.

This relaxation in the approximation requirement allows us to achieve an optimal stretch compact roundtrip routing scheme, as presented in the following theorem.

▶ **Reminder of Theorem 2.** *Let $G = (V,E,w)$ be a weighted undirected graph. Let $k \geq 1$ be an integer. There is a $(2k-1)$-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(n^{1/k})$, vertex labels of size $\tilde{O}(k)$ and packet headers of size $\tilde{O}(k)$.*

First, we describe the preprocessing algorithm, which computes the routing tables and assigns vertex labels. The input to the algorithm is a graph $G = (V,E,w)$ and an integer parameter $k > 1$. The algorithm uses Lemma 5 to build a hierarchy of vertex sets $A_0, A_1, \ldots, A_k$, where $A_0 = V$, $A_k = \emptyset$, $A_{i+1} \subseteq A_i$, $|A_i| = n^{1-i/k}$ and $|B_i(u)| = \tilde{O}(n^{1/k})$, for every $0 \leq i \leq k-1$. Next, for every $u \in V$, the preprocessing algorithm computes $B(u)$ and $C(u)$. Then, for every $u \in V$, the algorithm sets the routing table $\texttt{RT}(u)$ to:

$$\texttt{RT}(u) = \{\texttt{L}(u, T(C(v))) \mid v \in B(u)\},$$

and the label $\texttt{L}(u)$ to:

$$\texttt{L}(u) = \{\texttt{L}(u, T(C(p_i(u)))) \mid i \in [k]\}.$$

We now turn to bound the size of the routing tables and the vertex labels.

▶ **Lemma 14.** *$|\texttt{RT}(u)| = \tilde{O}(n^{1/k})$, and $|\texttt{L}(u)| = \tilde{O}(k)$, for every $u \in V$.*

Proof. From Lemma 5 it follows that $|B(u)| = \tilde{O}(n^{1/k})$. From Lemma 11 it follows that for every tree T it holds that $|\texttt{L}(u,T)| = \tilde{O}(1)$. Therefore, $|\texttt{RT}(u)| = |B(u)| \cdot \tilde{O}(1) = \tilde{O}(n^{1/k})$, as required. The label of each vertex is composed of k tree labels $L(u, T(C(p_i(u))))$ for every $i \in [k]$. From Lemma 11 we know that $|L(u, T(C(p_i(u))))| = \tilde{O}(1)$. Therefore, $|\texttt{L}(u)| = k \cdot \tilde{O}(1) = \tilde{O}(k)$, as required. ◀

The routing algorithm routes a message from u to v as follows. Let $\ell(x,y) = \min\{i \mid p_i(y) \in B(x)\}$ and let $\ell = \ell(u,v)$. We route from u to v on the shortest path between u and v in $T(C(p_\ell(v)))$. In figure 1 we show the roundtrip route between u and v according to this routing algorithm.

Next, we show that for every w on the shortest path from u to v in $T(C(p_\ell(v)))$ we have that $\texttt{L}(w, T(C(p_\ell(v)))) \in \texttt{RT}(w)$ and therefore w can route to v in $T(C(p_\ell(v)))$.

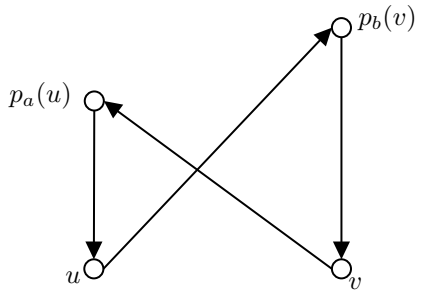


Figure 1 The roundtrip routing of Theorem 2. Let $a = \ell(v,u), b = \ell(u,v)$, where $\ell(x,y) = \min\{i \mid p_i(y) \in B(x)\}$. (We assume wlog that $b \geq a$.).

▶ **Lemma 15.** *For every $w \in P(u, p_\ell(v)) \cup P(p_\ell(v), v)$ it holds that $\mathtt{L}(w, T(C(p_\ell(v)))) \in \mathtt{RT}(w)$*

Proof. If $w \in P(u, p_\ell(v))$, then by Lemma 10, we know that $w \in C(p_\ell(v))$. Similarly, if $w \in P(p_\ell(v), v)$, then from Lemma 10, we know that $w \in C(p_\ell(v))$. Since $w \in C(p_\ell(v))$ it follows from the definition of $C(p_\ell(v))$ that $p_\ell(v) \in B(w)$. By the definition of $\mathtt{RT}(w)$ since $p_\ell(v) \in B(w)$ it follows that $\mathtt{L}(w, T(C(p_\ell(v)))) \in \mathtt{RT}(w)$, as required. ◀

We now turn to the main technical contribution of this section and prove that the stretch of the compact roundtrip routing scheme is $2k-1$.

▶ **Lemma 16.** $\hat{d}(u \leftrightarrow v) \leq (2k-1) \cdot d(u \leftrightarrow v)$

Proof. Let $u, v \in V$, let $b = \ell(u,v) = \min\{i \in [k] \mid p_i(v) \in B(u)\}$, and let $a = \ell(v,u) = \min\{i \in [k] \mid p_i(u) \in B(v)\}$. Assume, without loss of generality, that $b \geq a$. See Figure 1 for an illustration.

In the routing phase, we route from u to v on the shortest path between u and v in $T(C(p_b(v)))$. Similarly, we route from v to u on the shortest path between v and u in $T(C(p_a(u)))$. Therefore,

$$\begin{aligned}\hat{d}(u,v) &= d_{T(C(p_b(v)))}(u,v) \leq d(u,p_b(v)) + d(p_b(v),v) \text{ and } \hat{d}(v,u)\\ &= d_{T(C(p_a(u)))}(u,v) \leq d(v,p_a(u)) + d(p_a(u),u).\end{aligned}$$

From the triangle inequality, we have $d(u,p_b(v)) \leq d(u,v) + d(v,p_b(v))$, therefore we get that:

$$\hat{d}(u,v) = d(u,p_b(v)) + d(p_b(v),v) \leq d(u,v) + d(v,p_b(v)) + d(p_b(v),v) = d(u,v) + d(v \leftrightarrow p_b(v))$$

By symmetry, we also get that $\hat{d}(v,u) \leq d(v,u) + d(u \leftrightarrow p_a(u))$. By definition $\hat{d}(u \leftrightarrow v) = \hat{d}(u,v) + \hat{d}(v,u)$. Therefore, we get:

$$\begin{aligned}\hat{d}(u \leftrightarrow v) &= \hat{d}(u,v) + \hat{d}(v,u)\\ &\leq d(u,v) + d(v \leftrightarrow p_b(v)) + d(v,u) + d(u \leftrightarrow p_a(u))\\ &= d(u \leftrightarrow v) + d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u)).\end{aligned}$$

To get that $\hat{d}(u \leftrightarrow v) \leq (2k-1)d(u \leftrightarrow v)$, we show that $d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u)) \leq 2(k-1)d(u \leftrightarrow v)$ in the following claim.

▷ Claim 16.1. $d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u)) \leq 2(k-1)d(u \leftrightarrow v)$

Proof. Let $\Delta_i^v = d(v \leftrightarrow p_i(v)) - d(v \leftrightarrow p_{i-1}(v))$, for every $0 < i < k$. Notice that for every $0 < j < k$, it holds that:

$$\sum_{i=1}^{j} \Delta_i^v = d(v \leftrightarrow p_j(v)) - d(v \leftrightarrow p_{j-1}(v)) + d(v \leftrightarrow p_{j-1}(v)) - \cdots + d(v \leftrightarrow p_1(v)) - d(v \leftrightarrow p_0(v))$$
$$= d(v \leftrightarrow p_j(v)).$$

Since we assume (wlog) that $b \geq a$, we have:

$$d(v \leftrightarrow p_b(v)) = \sum_{i=1}^{b} \Delta_i^v = \sum_{i=1}^{a} \Delta_i^v + \sum_{i=a+1}^{b} \Delta_i^v = d(v \leftrightarrow p_a(v)) + \sum_{i=a+1}^{b} \Delta_i^v. \tag{1}$$

From the definition of b and a, for every $0 < i < a \leq b$, we have that both $p_i(u) \notin B(v)$ and $p_i(v) \notin B(u)$. Therefore, by applying Lemma 8 we get:

$$d(u \leftrightarrow p_a(u)) \leq a \cdot d(u \leftrightarrow v) \quad \text{and} \quad d(v \leftrightarrow p_a(v)) \leq a \cdot d(u \leftrightarrow v). \tag{2}$$

For every $a \leq i < b$, since $p_i(v) \notin B(u)$, we can apply Lemma 9 to get that $\Delta_i^v < 2d(u \leftrightarrow v)$. Therefore, we get:

$$\sum_{i=a+1}^{b} \Delta_i^v \leq \sum_{i=a+1}^{b} 2d(u \leftrightarrow v) = 2(b-a)d(u \leftrightarrow v) \tag{3}$$

Using the above three inequalities, we get:

$$\begin{aligned} d(u \leftrightarrow p_a(u)) + d(v \leftrightarrow p_b(v)) &\overset{(1)}{=} d(u \leftrightarrow p_a(u)) + d(v \leftrightarrow p_a(v)) + \sum_{i=a+1}^{b} \Delta_i^v \\ &\overset{(2)}{\leq} a \cdot d(u \leftrightarrow v) + a \cdot d(u \leftrightarrow v) + \sum_{i=a+1}^{b} \Delta_i^v \\ &\overset{(3)}{\leq} a \cdot d(u \leftrightarrow v) + a \cdot d(u \leftrightarrow v) + (b-a) \cdot 2d(u \leftrightarrow v) \\ &= 2bd(u \leftrightarrow v) \leq 2(k-1)d(u \leftrightarrow v), \end{aligned}$$

where the last inequality follows from the fact that $b \leq k-1$. ◁

Finally, since we know that $\hat{d}(u \leftrightarrow v) \leq d(u \leftrightarrow v) + d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u))$, and from Claim 16.1 we have that $d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u)) \leq 2(k-1)d(u \leftrightarrow v)$ we get:

$$\begin{aligned} \hat{d}(u \leftrightarrow v) &\leq d(u \leftrightarrow v) + d(v \leftrightarrow p_b(v)) + d(u \leftrightarrow p_a(u)) \\ &\leq d(u \leftrightarrow v) + 2(k-1)d(u \leftrightarrow v) = (2k-1)d(u \leftrightarrow v), \end{aligned}$$

As required. ◀

Theorem 2 follows from Lemma 14, Lemma 15 and Lemma 16.

5 Extending Section 4 to directed graphs

In this section, we consider roundtrip routing in weighted directed graphs. One might wonder why the routing scheme of Theorem 2 does not apply to or cannot be adapted to directed graphs. The key issue lies in the fact that Lemma 10 holds only for undirected graphs. Without this lemma, the routing process fails in directed graphs. Specifically, in directed

graphs, there exist cases where $u \in C(v)$ and $w \in P(u,v)$, but $w \notin C(v)$. Consequently, even if we store the set $\{\mathtt{L}(u, T(C(v))) \mid v \in B(u)\}$, for every vertex u, we may not be able to route correctly from u to v when $v \in B(u)$.

To overcome this issue, we consider bounded hop diameter graphs, where the hop diameter is the maximum number of edges on a shortest path between any two vertices u, v in the graph, i.e. $D_{hop} = \max_{u,v \in V}(|P(u,v)|)$. In such a case, we can achieve the following theorem.

▶ **Reminder of Theorem 4.** *Let $G = (V, E, w)$ be a weighted directed graph. Let $k \geq 1$ be an integer. There is a $(2k-1)$-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(D_{hop} n^{1/k})$, vertex labels of size $\tilde{O}(D_{hop} k)$ and packet headers of size $\tilde{O}(D_{hop})$.*

Proof. The preprocessing algorithm is identical to that of Theorem 2, with one key modification: instead of setting $\mathtt{RT}(u) = \{\mathtt{L}(u, T(C(v))) \mid v \in B(u)\}$, we store $\mathtt{RT}(u) = \{P(u,v) \mid v \in B(u)\}$, where $P(u,v)$ is the entire path from u to v, and similarly instead of setting $\mathtt{L}(u) = \{p_i(u) \mid i \in [k]\}$ we store $\mathtt{L}(u) = \{P(p_i(u), u) \mid i \in [k]\}$.

In the routing algorithm at the source vertex u to a destination vertex v, after determining $\ell = \min\{i \mid p_i(v) \in B(u)\}$. The entire path $P(u, p_\ell(v)) \cup P(p_\ell(v), v)$ is attached to the header to ensure that intermediate vertices can route correctly. The remainder of the proof follows the same steps as in Section 4. ◀

References

1. Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In Phillip B. Gibbons and Micah Adler, editors, *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 20–24. ACM, 2004. doi:10.1145/1007912.1007916.
2. Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. *ACM Trans. Algorithms*, 4(3):37:1–37:12, 2008. doi:10.1145/1367064.1367077.
3. Baruch Awerbuch, Amotz Bar-Noy, Nathan Linial, and David Peleg. Improved routing strategies with succinct tables. *J. Algorithms*, 11(3):307–341, 1990. doi:10.1016/0196-6774(90)90017-9.
4. Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discret. Math.*, 5(2):151–162, 1992. doi:10.1137/0405013.
5. Ruoxu Cen and Ran Duan. Roundtrip spanners with $(2k-1)$ stretch. *CoRR*, abs/1911.12411, 2019. arXiv:1911.12411.
6. Ruoxu Cen, Ran Duan, and Yong Gu. Roundtrip spanners with (2k-1) stretch. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 24:1–24:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.24.
7. Shiri Chechik. Compact routing schemes with improved stretch. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 33–41. ACM, 2013. doi:10.1145/2484239.2484268.
8. Lenore Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001. doi:10.1006/JAGM.2000.1134.

9 Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing for digraphs. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 885–886. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315068.

10 Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *J. Algorithms*, 50(1):79–95, 2004. doi:10.1016/J.JALGOR.2003.08.001.

11 Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *J. Algorithms*, 46(2):97–114, 2003. doi:10.1016/S0196-6774(03)00002-6.

12 Pierre Fraigniaud and Cyril Gavoille. A space lower bound for routing in trees. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes - Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 65–75. Springer, 2002. doi:10.1007/3-540-45841-7_4.

13 Cyril Gavoille, Christian Glacet, Nicolas Hanusse, and David Ilcinkas. On the communication complexity of distributed name-independent routing schemes. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 418–432. Springer, 2013. doi:10.1007/978-3-642-41527-2_29.

14 Alina Harbuzova, Ce Jin, Virginia Vassilevska Williams, and Zixuan Xu. Improved roundtrip spanners, emulators, and directed girth approximation. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4641–4669. SIAM, 2024. doi:10.1137/1.9781611977912.166.

15 Avi Kadria and Liam Roditty. Compact routing schemes in undirected and directed graphs. *arXiv preprint arXiv:2503.13753*, 2025. doi:10.48550/arXiv.2503.13753.

16 Goran Konjevod, Andréa W. Richa, and Donglin Xia. Optimal-stretch name-independent compact routing in doubling metrics. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 198–207. ACM, 2006. doi:10.1145/1146381.1146412.

17 Kofi A. Laing. Name-independent compact routing in trees. *Inf. Process. Lett.*, 103(2):57–60, 2007. doi:10.1016/J.IPL.2007.02.015.

18 Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1374–1392. SIAM, 2018. doi:10.1137/1.9781611975031.91.

19 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989. doi:10.1145/65950.65953.

20 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005. doi:10.1007/11523468_22.

21 Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):29:1–29:17, 2008. doi:10.1145/1367064.1367069.

22 Liam Roditty and Roei Tov. New routing techniques and their applications. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 23–32. ACM, 2015. doi:10.1145/2767386.2767409.

23 Eli Stafford and Chunjiang Zhu. Improved sourcewise roundtrip spanners with constant stretch. In Weili Wu and Guangmo Tong, editors, *Computing and Combinatorics - 29th International Conference, COCOON 2023, Hawaii, HI, USA, December 15-17, 2023, Proceedings, Part I*, volume 14422 of *Lecture Notes in Computer Science*, pages 297–309. Springer, 2023. doi:10.1007/978-3-031-49190-0_21.

24 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001*, pages 1–10. ACM, 2001. doi:10.1145/378580.378581.

25 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

A 7-stretch directed roundtrip routing with $|\mathtt{RT}(u)| = O(n^{1/3})$

In this section, we consider roundtrip routing in general weighted directed graphs. Roditty et. al [21] obtained a roundtrip routing scheme with $(4k+\varepsilon)$-stretch and routing tables of size $\tilde{O}(\frac{1}{\varepsilon}n^{1/k}\log W)$. If we set $k=3$ we get a $(12+\varepsilon)$-stretch roundtrip routing scheme with routing tables of size $\tilde{O}(\frac{1}{\varepsilon}n^{1/3}\log W)$. In this section, we present a 7-stretch roundtrip routing scheme with routing tables of size $\tilde{O}(n^{1/3})$. We prove:

▶ **Reminder of Theorem 3.** *Let $G=(V,E,w)$ be a weighted directed graph. There is a 7-stretch compact roundtrip routing scheme that uses local routing tables of size $\tilde{O}(n^{1/3})$, vertex labels of size $\tilde{O}(1)$ and packet headers of size $\tilde{O}(1)$.*

First, we describe the preprocessing algorithm, which computes the routing tables and assigns vertex labels. The input to the algorithm is a graph $G=(V,E,w)$. The algorithm uses Lemma 6 and Lemma 5 to build a hierarchy of vertex sets $V=A_0\supseteq A_1\supseteq A_2\supseteq A_3=\emptyset$, where $|A_i|=n^{1-i/k}$. For every $u\in V$ it holds that $|C(u,A_1)|=\tilde{O}(n^{1/3})$ and $|B_i(u)|=\tilde{O}(n^{1/k})$, where $0\le i\le 3$. Next, for every $u\in V$, the preprocessing algorithm computes $B(u)$ and $C(u)$. Then, for every $u\in V$, the algorithm sets the routing table $\mathtt{RT}(u)$ to:

$$\mathtt{RT}(u)=\begin{cases} L(w,T(B_0(u))) & \text{for every } w\in B_0(u)\\ L(u,T(B_0(w))) & \text{for every } w \text{ s.t. } u\in T(B_0(w))\\ L(u,T(C(w))) & \text{for every } w\in A_2\\ L(u,T(C(w))) & \text{for every } w\in B_1(u), \text{ if } P(u,w)\subseteq C(w)\\ L(w,T(C(p_2(z)))) & \text{for every } w\in B_1(u), z=\arg\min_{x\in P(u,w),x\notin C(w)}\{d(u,x)\}\end{cases}$$

and the label $\mathtt{L}(u)$ to:

$$\mathtt{L}(u)=\{\mathtt{L}(u,T(B_0(u))),\mathtt{L}(p_1(u),T(B_0(u))),\mathtt{L}(u,T(C(p_2(u))))\}$$

We now bound the size of the routing tables and the vertex labels.

▶ **Lemma 17.** *$|\mathtt{RT}(u)|=\tilde{O}(n^{1/3})$ and $|\mathtt{L}(u)|=\tilde{O}(1)$*

Proof. From Lemma 5 it follows that $|B_i(u)|=\tilde{O}(n^{1/3})$, for every $1\le i\le 2$. In addition we get that $|A_2|=\tilde{O}(n^{1/3})$. From Lemma 6 it follows that $|B_0(u)|=\tilde{O}(n^{1/3})$ and $|C(u,A_1)|=\tilde{O}(n^{1/3})$. From Lemma 11 it follows that for every tree T it holds that $|\mathtt{L}(u,T)|=\tilde{O}(1)$. Therefore, $|\mathtt{RT}(u)|\le\tilde{O}(1)\cdot(|B_0(u)|+|C(u,A_1)|+|A_2|+|B_1(u)|+|B_1(u)|)=\tilde{O}(n^{1/3})$, as required.

The label of each vertex is composed of three tree labels. From Lemma 11 it follows that each tree label is of size $\tilde{O}(1)$. Therefore, $|\mathtt{L}(u)|=3\cdot\tilde{O}(1)=\tilde{O}(1)$, as required. ◀

Algorithm 1 `Route`(u, v).

```
1  if v ∈ B_0(u) then Route from u to v on T(B_0(u)) ;
2  if v ∈ C(u, A_1) then Route from u to v on T(B_0(v)) ;
3  if p_1(v) ∈ B_1(u) then
4      if P(u, p_1(v)) ⊆ C(p_1(v)) then
5          Route from u to p_1(v) on T(C(p_1(v))) and then route from p_1(v) to v on
             T(B_0(v)).
6      else
7          z ← arg min_{x∈P(u,p_1(v)),x∉C(p_1(v))}{d(u, x)};
8          Route from u to p_1(v) on T(C(p_2(z))) and then route from p_1(v) to v on
             T(B_0(v)).
9  else
10     Route from u to v on T(C(p_2(v))).
```

The routing algorithm routes a message from u to v as follows. If $v \in B_0(u)$, then the algorithm routes the message using the tree $T(B_0(u))$. Otherwise, if $v \in C(u, A_1)$ then the algorithm routes the message using the tree $T(B_0(v))$. If $v \notin B_0(u)$ and $v \notin C(u, A_1)$, then the algorithm checks if $p_1(v) \in B_1(u)$. In this case, if $P(u, p_1(v)) \subseteq C(p_1(v))$ then the algorithm routes on $T(C(p_1(v)))$ from u to $p_1(v)$, and then from $p_1(v)$ to v on the tree $T(B_0(v))$ (see Figure 2 (a)). Otherwise, if $P(u, p_1(v)) \not\subseteq C(p_1(v))$, then the algorithm routes from u to $p_1(v)$ on the tree $T(C(p_2(z)))$, where $z = \arg\min_{x \in P(u,p_1(v)), x \notin C(p_1(v))}\{d(u, x)\}$, and then the algorithm routes from $p_1(v)$ to v on the tree $T(B_0(v))$ (see Figure 2 (b)).

Finally, if $p_1(v) \notin B_1(u)$, then the algorithm routes from u to v on the tree $T(C(p_2(v)))$ (see Figure 2 (c)). A pseudo-code for the routing algorithm is given in Algorithm 1.

In the next lemma, we show that all intermediate vertices have the necessary information to complete the routing process once the routing tree is determined.

▶ **Lemma 18.** *The following properties hold:*

1. *If $v \in B_0(u)$, then for every $w \in P(u, v)$, we have $\mathtt{L}(w, T(B_0(u))) \in \mathtt{RT}(w)$.*
2. *If $v \in C(u, A_1)$, then for every $z \in P(u, v)$, we have $\mathtt{L}(z, T(B_0(v))) \in \mathtt{RT}(z)$.*
3. *If $p_1(v) \in B_1(u)$ and $P(u, p_1(v)) \subseteq C(v)$, then for every $z \in P(u, p_1(v))$, we have $\mathtt{L}(z, T(C(p_1(v)))) \in \mathtt{RT}(z)$.*
4. *If $w \in A_2$, then for every $z \in P(u, w) \cup P(w, v)$, we have $\mathtt{L}(z, T(C(w))) \in \mathtt{RT}(z)$.*
5. *For every $w \in P(p_1(v), v)$, we have $\mathtt{L}(w, B_0(v)) \in \mathtt{RT}(w) \cup \mathtt{L}(v)$.*

Proof. From Lemma 7, we know that for every $v \in B(u, X)$ and $w \in P(u, v)$, we have that $w \in B(u, X)$. Therefore, properties 1 and 2 hold.

For property 3, we have that the entire path $P(u, p_1(v))$ is contained in $C(p_1(v))$. Hence, any sub-path of $P(u, p_1(v))$ is also in $C(p_1(v))$, and we have $\mathtt{L}(z, T(C(p_1(v)))) \in \mathtt{RT}(z)$ for every $z \in P(u, p_1(v))$.

For property 4, since $w \in A_2$, we know that $C(w) = V$. Therefore, all vertices in the graph (and specifically z) hold $\mathtt{L}(z, T(C(w))$.

For property 5, for $p_1(v)$, we have $\mathtt{L}(p_1(v), T(B_0(v))) \in \mathtt{L}(v)$. For every other $w \in P(p_1(v), v)$, we have from Lemma 13 that $v \in C(w)$ and therefore $w \in T(B(v))$, and we have $\mathtt{L}(w, T(B_0(v)))$ in $\mathtt{RT}(w)$, as required. ◀

We now turn to prove that the stretch of the compact roundtrip routing scheme is 7.

▶ **Lemma 19.** $\hat{d}(u \leftrightarrow v) \leq 7 \cdot d(u \leftrightarrow v)$

Proof. First, consider the case that $v \in B_0(u)$. In this case, the routing algorithm routes from u to v along $P(u, v)$, so $\hat{d}(u, v) = d(u, v)$. Since $v \in B_0(u)$, by the definition of $C(v, A_1)$, we know that $u \in C(v, A_1)$. Therefore, the routing algorithm from v to u follows $P(v, u$. Thus,

$$\hat{d}(u \leftrightarrow v) = \hat{d}(u, v) + \hat{d}(v, u) = d(u, v) + d(v, u) = d(u \leftrightarrow v),$$

as required. Similarly, if $v \in C(u, A_1)$, using symmetrical arguments, we get that $\hat{d}(u \leftrightarrow v) = d(u \leftrightarrow v)$, as required.

Next, consider the case that $v \notin B_0(u)$ and $v \notin C(u, A_1)$. In the following claim, we show that in this case, $\hat{d}(u, v) \leq d(u, v) + 3d(u \leftrightarrow v)$. Using this claim, we obtain:

$$\hat{d}(u \leftrightarrow v) = \hat{d}(u, v) + \hat{d}(v, u) \leq d(u, v) + 3d(u \leftrightarrow v) + d(v, u) + 3d(u \leftrightarrow v) \leq 7d(u \leftrightarrow v),$$

as required.

▷ Claim 19.1. If $v \notin B_0(u)$ and $v \notin C(u, A_1)$, then:

$$\hat{d}(u, v) \leq d(u, v) + 3d(u \leftrightarrow v).$$

Proof. Since $v \notin B_0(u)$ and $v \notin C(u, A_1)$, we know from the definitions of $B_0(u)$ and $C(u, A_1)$ that:

$$d(u \leftrightarrow p_1(u)) \leq d(u \leftrightarrow v) \quad \text{and} \quad d(v \leftrightarrow p_1(v)) \leq d(u \leftrightarrow v). \tag{1}$$

We divide the proof into two cases: the case that $p_1(v) \notin B(u)$ and the case that $p_1(v) \in B(u)$. If $p_1(v) \notin B(u)$ we can apply Lemma 9 and get:

$$d(v \leftrightarrow p_2(v)) \leq 2d(u \leftrightarrow v) + d(v \leftrightarrow p_1(v)) \overset{(1)}{\leq} 3d(u \leftrightarrow v). \tag{2}$$

Since $p_1(v) \notin B(u)$, the routing algorithm routes from u to v on the tree $T(C(p_2(v))$. Therefore, we have:

$$\begin{aligned}\hat{d}(u, v) &= d_{T(C(p_2(v)))}(u, v) \leq d(u, p_2(v)) + d(p_2(v), v) \\ &\overset{\triangle}{\leq} d(u, v) + d(v \leftrightarrow p_2(v)) \overset{(2)}{\leq} d(u, v) + 3d(u \leftrightarrow v),\end{aligned}$$

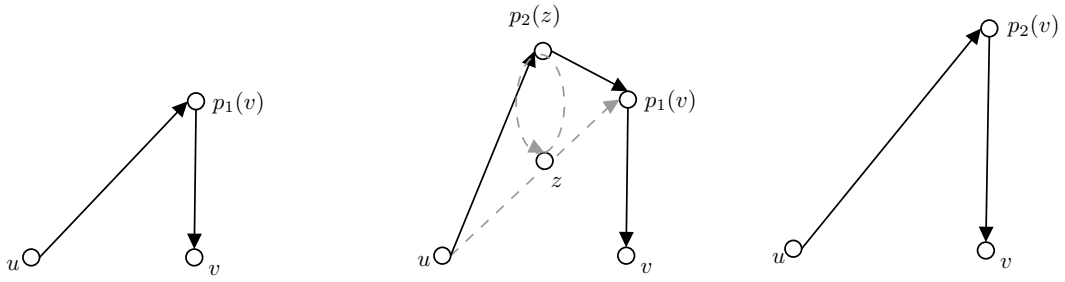
as required.

Next, consider the case that $p_1(v) \in B_1(u)$. If $P(u, p_1(v)) \subseteq C(p_1(v))$, then the routing algorithm routes along the shortest path from u to $p_1(v)$. Therefore, we have:

$$\hat{d}(u, v) \leq d(u, p_1(v)) + d(p_1(v), v) \overset{\triangle}{\leq} d(u, v) + d(v \leftrightarrow p_1(v)) \overset{(1)}{\leq} d(u, v) + d(u \leftrightarrow v),$$

as required.

Otherwise, if $P(u, p_1(v)) \not\subseteq C(p_1(v))$, let z be the first vertex in $P(u, p_1(v))$ such that $z \notin C(p_1(v))$. From the definition of $C(p_1(v))$, it follows that $p_1(v) \notin B_1(z$. From the definition of $B_1(z)$ we get that $d(z \leftrightarrow p_2(z)) \leq d(z \leftrightarrow p_1(v))$. Since $z \in P(u, p_1(v))$, it follows that $d(z \leftrightarrow p_1(v)) \leq d(u \leftrightarrow p_1(v))$. Combining these inequalities, we get that: $d(z \leftrightarrow p_2(z)) \leq d(u \leftrightarrow p_1(v))$.



(a) $p_1(v) \in B_1(u)$ $P(u, p_1(v)) \subseteq C(p_1(v))$

(b) $p_1(v) \in B_1(u)$ $z \notin C(p_1(v))$ $z \in P(u, p_1(v))$

(c) $p_1(v) \notin B_1(u)$

Figure 2 The routing of Theorem 3. $v \notin B_0(u) \cup C(u, A_1)$.

In this case, the routing algorithm routes from u to $p_1(v)$ on the tree $T(C(p_2(z))$, and then from $p_1(v)$ to v. We get that:

$$\begin{aligned}
\hat{d}(u,v) &= d_{T(C(p_2(z)))}(u, p_1(v)) + d(p_1(v), v) \\
&\leq d(u, p_2(z)) + d(p_2(z), p_1(v)) + d(p_1(v), v) \\
&\overset{\triangle}{\leq} d(u, z) + d(z, p_2(z)) + d(p_2(z), z) + d(z, p_1(v)) + d(p_1(v), v) \\
&= d(u, z) + d(z \leftrightarrow p_2(z)) + d(z, p_1(v)) + d(p_1(v), v).
\end{aligned}$$

Since $z \in P(u, p_1(v))$, we know that $d(u, z) + d(z, p_1(v)) = d(u, p_1(v))$. Recall that $d(z \leftrightarrow p_2(z)) \leq d(u \leftrightarrow p_1(v))$. Therefore:

$$\begin{aligned}
\hat{d}(u,v) &\leq d(u, z) + d(z, p_1(v)) + d(p_1(v), v) + d(z \leftrightarrow p_2(z)) \\
&\leq d(u, p_1(v)) + d(p_1(v), v) + d(u \leftrightarrow p_1(v)) \\
&\overset{\triangle}{\leq} d(u, v) + d(v \leftrightarrow p_1(v)) + d(u \leftrightarrow v) + d(v \leftrightarrow p_1(v)) \\
&\overset{(1)}{\leq} d(u, v) + d(u \leftrightarrow v) + d(u \leftrightarrow v) + d(u \leftrightarrow v) = d(u, v) + 3d(u \leftrightarrow v),
\end{aligned}$$

as required. ◁

From Claim 19.1, we have that $\hat{d}(u, v) \leq d(u, v) + 3d(u \leftrightarrow v)$ and $\hat{d}(v, u) \leq d(u, v) + 3d(u \leftrightarrow v)$. Therefore:

$$\hat{d}(u \leftrightarrow v) = \hat{d}(u, v) + \hat{d}(v, u) \leq d(u, v) + 3d(u \leftrightarrow v) + d(v, u) + 3d(u \leftrightarrow v) = 7d(u \leftrightarrow v),$$

as required. ◀

Theorem 3 follows from Lemma 17, Lemma 18 and Lemma 19.

TEE Is Not a Healer: Rollback-Resistant Reliable Storage

Sadegh Keshavarzi
University of Surrey, Guildford, UK

Gregory Chockler
University of Surrey, Guildford, UK

Alexey Gotsman
IMDEA Software Institute, Madrid, Spain

Abstract

Recent advances in secure hardware technologies, such as Intel SGX or ARM TrustZone, offer an opportunity to substantially reduce the costs of Byzantine fault-tolerance by placing the program code and state within a secure enclave known as a *Trusted Execution Environment (TEE)*. However, the protection offered by a TEE only applies during program execution. Once power is switched off, the non-volatile portion of the program state becomes vulnerable to *rollback* attacks wherein it is undetectably reverted to an older version. In this paper we consider the problem of implementing reliable read/write registers out of failure-prone replicas subject to state rollbacks. To this end, we introduce a new unified model that captures multiple failure types that can affect a TEE-based system and establish tight bounds on the fault-tolerance of register constructions in this model. We consider both the static case, where failure thresholds hold throughout the entire execution, and the dynamic case, where any number of replicas can roll back, provided these failures do not occur too often. Our dynamic register emulation algorithm, TEE-Rex, provides the first correct implementation of a distributed state recovery procedure that requires neither durable storage nor specialized hardware, such as trusted monotonic counters.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Trusted execution environments, fault tolerance, crash recovery

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.39

Related Version *Extended Version*: `https://arxiv.org/abs/2505.18648` [26]

Funding This work was partially supported by the projects BYZANTIUM and DECO funded by MCIN/AEI, and REDONDA funded by the CHIST-ERA network.

1 Introduction

Tolerating Byzantine failures plays a major role in designing modern reliable distributed systems, and in particular blockchains. Unfortunately, tolerating Byzantine failures is expensive in terms of message complexity [15], latency [2, 43], and the number of replicas required [16, 32, 39]. Recent advances in secure hardware technologies, such as Intel SGX or ARM TrustZone, offer an opportunity to substantially reduce these costs by placing the program's code and a portion of its state in a secure enclave known as a *Trusted Execution Environments (TEE)*. Applications can furthermore detect any runtime alterations to the state stored outside the enclave using integrity metadata stored inside it. These mechanisms together can then be used to convert Byzantine failures to simple crash or omission failures, which are less expensive to tolerate [7, 31, 41, 42].

However, a TEE is a not a panacea, because the protection it offers only applies during program execution. Once power is switched off, the non-volatile portion of the program state becomes vulnerable to tampering by an attacker. To address this, hardware manufacturers


© Sadegh Keshavarzi, Gregory Chockler, and Alexey Gotsman;
licensed under Creative Commons License CC-BY 4.0



provide a *sealing* mechanism [12] that allows programs to encrypt a portion of their state using a device-specific key before storing it in non-volatile storage. When the machine restarts, the TEE can check whether the sealed state was indeed written by it in the past. However, it cannot determine whether this state is the most recent version. This limitation makes TEE-based systems vulnerable to *rollback* attacks where the sealed state is undetectably replaced with an older version [8, 13, 35, 40, 42].

In this paper we study theoretical foundations for building reliable distributed systems in the presence of rollbacks. We start by introducing *crash-restart-rollback* (CRR), a new unified failure model that formalizes rollbacks alongside other types of failures that can occur in a TEE-based system. In this model, processes can use local non-volatile storage to persist their states and are assumed to behave correctly while they are executing. A CRR-faulty process can permanently crash, crash and restart infinitely often, or crash and restart with the content of its non-volatile storage being undetectably reverted to an older version.

We then study the problem of implementing reliable read/write registers in an asynchronous message-passing system where the register's state is stored at a collection of n CRR failure-prone *replicas* accessed by arbitrarily many crash-prone *clients*. We consider two sub-classes of the above CRR failure model for replicas – *static* and *dynamic*. Static failure models assume the existence of a priori fixed thresholds restricting the number of replicas that can experience failures of various types in every execution; this is similar to the classical crash or Byzantine failure models. The dynamic failure models generalize the static ones by permitting any number of replicas to roll back in exchange for requiring that these failures do not occur too often.

Static failure models. For the static failure models, we give a fine-grained characterization of failure resilience, time complexity, and non-volatile storage requirements of a register implementation in terms of several thresholds: k on the total number of any CRR failures; $r \leq k$ on the number of rollback failures; and $b \leq n$ on the number of correct replicas that can crash at least once, but eventually stay up. Our main result establishes the following: a wait-free atomic multi-writer/multi-reader (MWMR) register can be implemented in the static model if and only if $n \geq 2k + \min(b, r) + 1$. Our algorithm matching this bound has the latency of 4 message delays for both writes and reads, which is the same as for crash fault-tolerant register implementations [11, 33] and is thus optimal. This stands in a sharp contrast with the Byzantine-resilient register constructions, where write and read latencies of 4 and 8 message delays are inherent even for implementing a single-writer/multi-reader (SWMR) wait-free atomic register [14].

Our lower bound implies that, when $r \leq b$, rollback failures are no different from Byzantine failures in terms of failure resilience: the number of replicas $n \geq 2k + r + 1$ required in this case is the same as that needed to implement Byzantine-resilient registers when at most r out of k faulty replicas can behave arbitrarily [18]. On the other hand, the two models are separated when $b < r$: in this case, a register can be implemented in CRR with $2k + b + 1$ replicas, which is strictly fewer than $2k + r + 1$ required by Byzantine fault-tolerant implementations.

We also establish a lower bound on non-volatile storage consumption: when $2k + r + 1 \leq n < 2k + b + 1$, at least $2k + r + 1$ replicas must store their states persistently on their local non-volatile storage. Since this storage is not protected by TEEs, in practice applications must rely on expensive mechanisms to monitor its integrity at runtime, e.g., using TEE-protected integrity metadata, such as Merkle trees. Our results show that for some resilience levels these overheads are unavoidable.

Dynamic failure models. For the dynamic failure models, we first show that no implementation of a single-writer/single-reader (SWSR) safe register can use fewer than $d + c + 1$ replicas, where d bounds the number of replicas that either crash permanently or never stop crashing and recovering, and $c \geq d$ bounds the number of replicas that crash at least once. While it is well-known that no register implementation can exist without a majority of replicas being eventually up ($n \geq 2d + 1$), our result further refines this lower bound for the case when $c > d$.

For a matching upper bound, we propose an algorithm, called TEE-Rex, that implements an MWMR atomic register using $n \geq d + c + 1$ replicas. The algorithm is always safe and is wait-free in executions where: $n - d$ replicas eventually stop crashing; and either replicas do not crash too often or $n - c$ replicas never crash at all. Unlike in the static model, these conditions place no restrictions on the number of replicas that can roll back. The algorithm is parameterized by d, but is unaware of c. It achieves the optimal time complexity of 4 message delays for both writes and reads in failure-free executions.

The most interesting ingredient of TEE-Rex is a novel distributed recovery protocol that enables crashed replicas to rebuild their state and become fully operational upon restart. As we explain in §4.1, implementing recovery correctly is nontrivial: a naive approach of simply querying a read quorum and adopting the state of the most up-to-date replica is fundamentally unsafe [36]. In fact, as we discovered in this work, some of the prior recoverable register implementations suffer from nontrivial safety bugs [13, 17] (§5). Others do not guarantee liveness [35, 42], or require trusted hardware-based primitives such as persistent monotonic counters [25, 36]. In contrast, our TEE-Rex algorithm provides what we believe is the first self-contained and rollback-resilient construction of a read/write register under dynamic failure models.

2 System Model

We consider an asynchronous message-passing system consisting of a collection of failure-prone processes $\mathcal{P} = \{p_1, p_2, \ldots\}$ implementing a high-level object abstraction. The set of processes is partitioned into a set $\mathcal{R} = \{p_1, \ldots, p_n\}$ of $n > 1$ *replicas*, and a set $\{p_{n+1}, p_{n+2}, \ldots\}$ of possibly infinitely many *clients*. Similarly to the classical faulty shared memory model of [3, 24], the replicas are responsible for storing the object state, and the clients interact with the replicas to handle the requests supported by the implemented object type. To study tradeoffs between failure resilience and non-volatile storage consumption, we assume that $s \geq 0$ replicas have access to non-volatile storage.

Formally, an object *implementation* is a composition of client and replica automata. An execution of an implementation is a sequence of states interleaved with atomic send and receive actions starting from an initial state. The action atomicity ensures that all non-volatile storage modifications performed as part of the received message handling are *failure atomic* [9], i.e., their effects are either persisted in their entirety or not at all.

Failure models. A failure model is a predicate over executions. An execution α is *valid under a failure model* $\mathcal{F}$ (or simply $\mathcal{F}$*-valid*) if $\mathcal{F}(\alpha)$ holds. For failure models $\mathcal{F}$ and $\mathcal{F}'$, $\mathcal{F} \preceq \mathcal{F}'$ iff $\forall \alpha.\, \mathcal{F}(\alpha) \implies \mathcal{F}'(\alpha)$.

Client failures. Clients can experience permanent *crash* failures, but otherwise do not deviate from their prescribed protocols. A client is *correct* in an execution if it never crashes, and is *faulty*, otherwise.

Replica failures. Our baseline failure model for replicas, to which we refer as *crash-restart-rollback* (CRR), formalizes the types of failures that can occur in a TEE-based system (see §1). Under CRR, a replica can experience a *crash* that interrupts its execution either permanently or temporarily. In the former case, the replica stops taking steps; in the latter case, it *restarts*, by executing a specified **on restart** block and then continues its execution from the state reached thereupon. For simplicity, we assume that when a replica is restarted, the restart and its immediately preceding crash occur *simultaneously* as a single atomic event. We refer to the restarted replicas that have completed their **on restart** blocks as *active*. Upon restart, a replica with non-volatile storage may additionally experience a *rollback* failure, which causes the content of its non-volatile storage to revert to a version older than it had before the latest crash.

A replica is *up* in an interval $[t, t']$ if it does not crash without restart before t, and does not crash (either with or without restart) at all times in $[t, t']$; a replica is *up after* t, if it is up in $[t, \infty)$; a replica p_i is *eventually up* if there exists t such that p_i is up after t. Replicas in CRR fall into four disjoint classes:

- a replica is *perfect* if it never crashes;
- a replica is *benign* if it crashes and restarts at least once, but is eventually up, and furthermore, it never rolls back;
- a replica is *crash-faulty* if it crashes without restarting or crashes and restarts infinitely often, and furthermore, it never rolls back; and
- a replica is *rollback-faulty* if it rolls back at least once.

Perfect and benign replicas are together called *correct*; crash-faulty and rollback-faulty replicas are together called *faulty*.

Note that CRR generalizes the classical crash-recovery failure model [22] by extending its set of faulty behaviors with rollback failures. It is also strictly weaker than the Byzantine failure model since faulty replicas are required to follow their prescribed protocols. The relationship of our model to other prior failure models is further discussed in §5.

Channel reliability assumptions. We assume that every pair of processes p_i and p_j are connected via point-to-point authenticated links such that if both p_i and p_j are up after some t, then every message sent by p_i to p_j after t is eventually delivered by p_j. Note that this assumption is strictly weaker than the standard notion of a reliable channel. Specifically, it does not require reliability for messages sent by a correct process to another correct process before both processes stop crashing.

Specifications. We use standard safety and liveness notions to specify correctness conditions for register implementations: atomicity [28] (linearizability [21]), safeness [28], wait-freedom [19], and obstruction-freedom [20]. We consider both *single-writer/single-reader (SWSR)* and *multiple-writer/multiple-reader (MWMR)* register implementations.

Fault-tolerant implementations. An object implementation is *safe under a failure model* $\mathcal{F}$ (or simply $\mathcal{F}$*-safe*) if it satisfies safety in all $\mathcal{F}$-valid executions. We define $\mathcal{F}$*-liveness* similarly. An implementation is $\mathcal{F}$*-tolerant* if it is both safe and live under $\mathcal{F}$.

We study the implementability of registers under failure models where, in every execution, arbitrarily many clients can crash and replicas are subject to a restricted variant of the baseline CRR model above. Specifically, in §3 we consider *static* failure models $\mathsf{CRR}(k, r, b)$ for replicas where the number of faulty, rollback-faulty, and benign replicas in every execution is bounded by $k \leq n$, $r \leq k$, and $b \leq n$, respectively. Note that $\forall k, r, b.\, \mathsf{CRR}(k, r, b) \preceq \mathsf{CRR}$. In §4 we consider more flexible *dynamic* failure models.

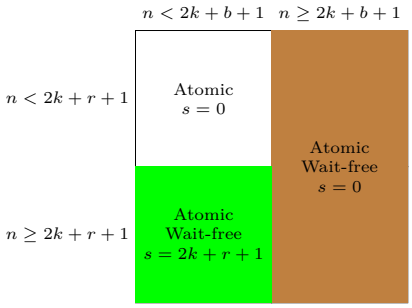


Figure 1 Resilience (k, r, b) and non-volatile storage usage (s) of the upper bound. Each square corresponds to the chunk of the problem space where the conjunction of the conditions on the axes holds.

3 Register Implementations in Static Failure Models

In this section we give a full characterization of the costs of implementing a register under static failure models in terms of its failure resilience, time complexity, and non-volatile storage requirements. For resilience, the following theorem establishes a tight bound on the total number of replicas n as a function of thresholds on the number of replicas of different types.

▶ **Theorem 1.** *Assume that all replicas have access to non-volatile storage. Then for all k, r, b, there exists a $\mathsf{CRR}(k, r, b)$-tolerant implementation of a wait-free atomic MWMR register if and only if $n \geq 2k + \min(b, r) + 1$.*

When $r = 0$, the bound in the theorem specializes to $n \geq 2k + 1$, the same as for crash fault-tolerant register implementations. When $r \leq b$, the bound specializes to $n \geq 2k + r + 1$. This matches the known generalization [18] of the $n \geq 3k + 1$ Byzantine implementability bound for registers [14, 34] to the case when at most r out of k faulty replicas can behave arbitrarily and the rest can crash but not deviate from the prescribed protocol. On the other hand, if $b < r$, then the theorem shows that we can implement a $\mathsf{CRR}(k, r, b)$-tolerant wait-free atomic register with $n = 2k + b + 1 < 2k + r + 1$ replicas – strictly fewer than in the Byzantine case. We next refine Theorem 1 to additionally give a tight bound on the number of replicas that must be equipped with non-volatile storage.

▶ **Theorem 2.** *Let s be the number of replicas with non-volatile storage. Then for all k, r, b, there exists a $\mathsf{CRR}(k, r, b)$-tolerant implementation of a wait-free atomic MWMR register if and only if*

$$(2k + r + 1 \leq n < 2k + b + 1 \wedge s \geq 2k + r + 1) \vee (n \geq 2k + b + 1).$$

Note that the constraint on n in Theorem 2 implies that in Theorem 1; we illustrate the constraint in Figure 1. The above theorem shows that, when $n \geq 2k + b + 1$, atomic registers can be implemented without any stable storage at all. In practice, avoiding non-volatile storage offers substantial performance gains when the register state fits within the TEE-protected RAM, as it avoids the overheads of monitoring storage integrity at runtime.

Finally, the result in Theorem 2 can be strengthened along several dimensions:

- The time complexity of the upper bound we present in §3.1 is always 4 message delays for both writes and reads. This matches the time complexity of crash fault-tolerant implementations, which is optimal. This property stands in a sharp contrast with Byzantine-resilient register constructions, where write and read latencies of 4 and 8 message delays, respectively, are inherent even for single-writer/multi-reader (SWMR) wait-free atomic registers [14].

- The lower bound in Theorem 2 is established for obstruction-free safe implementations of SWSR registers.
- When either of b or r is unknown, a tight bound is obtained from Theorem 2 by letting $b \triangleq n$ or $r \triangleq k$.
- If the number of replicas is less than the resilience bound ($n < 2k + \min(b, r) + 1$), the upper bound does not sacrifice safety, but only liveness (the top-left box in Figure 1). In §4 we leverage this property to develop a register construction under dynamic failure models that do not require all failure thresholds to hold for the entire execution.

Our findings collectively demonstrate that, while existing Byzantine fault-tolerant register constructions [1,14,18,34] can be used for implementing $\mathsf{CRR}(k, r, b)$-tolerant atomic registers, they are not well-suited for this purpose due to their sub-optimal failure resilience and latency. We next present an algorithm that gives a witness for our upper bound; we defer the proofs of the other results to [26, §A].

3.1 Upper Bound

An algorithm $\mathcal{A}$ that shows our upper bound for $\mathsf{CRR}(k, r, b)$ is presented in Figure 2. It is based on the MWMR variant of ABD [5,11,33], where read and write quorums are sets of replicas of size q_r and q_w, respectively. The algorithm uses the truth value of the predicate $P(n, k, r, b) = (2k + r + 1 \leq n < 2k + b + 1)$ to determine the values of q_r and q_w (line 2), the restart handler logic (lines 6–7), and whether non-volatile storage should be used to store the replica's state (line 5). In the following we refer to the variant of $\mathcal{A}$ executed if P holds as algorithm $\mathcal{A}_1$, and as algorithm $\mathcal{A}_2$ otherwise. The former corresponds to the bottom-left box in Figure 1, while the latter corresponds to the remaining portions of the figure.

Replica states. Each replica stores a copy of the register state in a state variable (line 3), which is a tuple consisting of the register value state.val and timestamp state.ts. Timestamps are pairs of a counter and the client identifier, ordered lexicographically: $(cnt_1, j_1) \leq (cnt_2, j_2)$ iff $cnt_1 < cnt_2 \vee (cnt_1 = cnt_2 \wedge j_1 \leq j_2)$. Each replica also maintains a Boolean flag stale: if this flag is true, the replica is not allowed to respond to read requests.

Quorum-access functions. The clients use auxiliary functions `read_quorum` (lines 14–19) and `write_quorum` (lines 29–34) to respectively query and update various components of replica states at a quorum. For `read_quorum`, the *req* argument specifies the state elements to query: e.g., TSVal corresponds to (state.ts, state.val). For `write_quorum`, *req* specifies which state elements to modify and their new values: e.g., $\mathsf{TSVal}(ts, v)$ to set (state.ts, state.val) to (ts, v).

The `read_quorum` (respectively, `write_quorum`) function starts by generating a globally unique request identifier *id* (lines 15 and 30); in practice this can be implemented using cryptographic nonces. The function then broadcasts a $\texttt{READ}(id, req)$ (respectively, $\texttt{WRITE}(id, req)$) message to all replicas, and awaits `READ_ACK` (respectively, `WRITE_ACK`) messages from q_r (respectively, q_w) replicas. Since channels may fail to deliver messages sent before their respective destinations stop crashing (§2), to ensure liveness, both `read_quorum` and `write_quorum` continue retransmitting requests until receiving the desired quorums of responses. The `read_quorum` function then returns the set of payloads received in `READ_ACK` messages from a read quorum of replicas.

Whenever a replica with stale = FALSE receives a $\texttt{READ}(id, req)$ message (line 35), it responds with a `READ_ACK` message, carrying *id* and the current value of the state variable associated with *req*. Whenever a replica receives a $\texttt{WRITE}(id, req)$ message (line 20), it applies

Predicates and constants:

1 $P(n,k,r,b) \triangleq 2k+r+1 \leq n < 2k+b+1$

2 $(q_w, q_r) \triangleq (n-k, n-k)$ if $P(n,k,r,b)$; and $(n-k, k+1)$ otherwise

State:

3 state, initially $((0,0), v_0)$, with selectors ts and val

4 stale $\in \{\text{TRUE}, \text{FALSE}\}$, initially FALSE

5 state, stale: **non-volatile** if $P(n,k,r,b) \wedge (i \leq 2k+r+1)$

6 **on restart**

7 ⎿ **if** $\neg P(n,k,r,b)$ **then** stale $\leftarrow$ TRUE

8 **function** `write`(v)

9 $S \leftarrow$ `read_quorum`(TS)

10 $cnt \leftarrow \max\{cnt' \mid (cnt', _) \in S\}$

11 $ts \leftarrow (cnt+1, i)$

12 `write_quorum`(TSVal(ts, v))

13 **return** ack

14 **function** `write_quorum`(req)

15 $id \leftarrow$ `get_unique_id`()

16 **do**

17 ⎿ **send** `WRITE`(id, req) **to** $\mathcal{R}$

18 **periodically until received**
$\{$`WRITE_ACK`$(id, j) \mid p_j \in Q\}$
for some Q **such that** $|Q| \geq q_w$

19 **return**

20 **when received** `WRITE`$(id,$ TSVal$(ts,v))$ **from** p_j

21 **if** $ts >$ state.ts **then**

22 ⎿ (state.ts, state.val) $\leftarrow (ts, v)$

23 ⎿ **send** `WRITE_ACK`(id, i) **to** p_j

24 **function** `read`()

25 $S \leftarrow$ `read_quorum`(TSVal)

26 **let** (ts, v) **be such that** $(ts, v) \in S \wedge$
$ts = \max\{ts' \mid (ts', _) \in S\}$

27 `write_quorum`(TSVal(ts, v))

28 **return** v

29 **function** `read_quorum`(req)

30 $id \leftarrow$ `get_unique_id`()

31 **do**

32 ⎿ **send** `READ`(id, req) **to** $\mathcal{R}$

33 **periodically until received**
$\{$`READ_ACK`$(id, x_j, j) \mid p_j \in Q\}$
for some Q **such that** $|Q| \geq q_r$

34 **return** $\{x_j \mid p_j \in Q\}$

35 **when received** `READ`(id, req) **from** p_j

36 **pre:** stale = FALSE

37 $r \leftarrow$ **case** req **do**

38 TS: state.ts

39 TSVal: (state.ts, state.val)

40 ⎿ **send** `READ_ACK`(id, r, i) **to** p_j

Figure 2 Pseudocode for algorithm $\mathcal{A}$ at process $p_i \in \mathcal{P}$.

the update encoded in req to its state and acknowledges this fact with a `WRITE_ACK` message. In particular, if $req =$ TSVal(ts, v) and state.ts $< ts$, then (state.ts, state.val) is set to (ts, v); otherwise, the state is left unchanged.

Read and write protocols. To write a value v, a process p_i first calls `read_quorum` to retrieve a set of timestamps from a read quorum (line 9). It then selects the highest timestamp counter cnt and calls `write_quorum` to store v with timestamp $(cnt+1, i)$ at a write quorum. To read a value, a process p_i first calls `read_quorum` to retrieve a set of timestamp-value pairs from a read quorum (line 25). It then selects the pair (ts, v) with the highest timestamp, calls `write_quorum` to store it at a write quorum, and returns v.

Algorithm $\mathcal{A}_1$. Algorithm $\mathcal{A}_1$ handles the case when the predicate $P(n,k,r,b)$ holds, i.e., $2k+r+1 \leq n < 2k+b+1$ (the bottom-left box in Figure 1 and the first disjunct in Theorem 2). The algorithm uses read and write quorums of size $q_w = q_r = n-k$ (line 2) and keeps the state of $2k+r+1$ replicas in non-volatile storage (line 5), which is the minimum required by the lower bound of Theorem 2. Its restart handler returns immediately (lines 6–7), so the stale flag is always FALSE at all replicas, which can thus respond to `READ` messages.

Since in every execution at least $n - k > 0$ replicas are eventually up, both write and read quorums are always available, and, as a result, algorithm $\mathcal{A}_1$ is wait-free. We defer the full proof of its linearizability to [26, §A] and only prove the following key property the proof relies on:

▶ **Property 3** (Real-Time Order). *Let WQ be a completed call to* `write_quorum`(TSVal$(x, _)$)*, and RQ be a completed call to* `read_quorum`(TS) *or* `read_quorum`(TSVal)*. If RQ is invoked after the completion of WQ, then it returns a set containing a value $\geq x$ for* state.ts*.*

Proof of Property 3 for $\mathcal{A}_1$. Since WQ returns, it has executed the TSVal$(x, _)$ request at $q_w = n - k$ replicas. Since RQ returns, it has retrieved the value of state.ts from a $q_r = k + 1$ replicas. The intersection of the two quorums has a cardinality $\geq (n-k)+(n-k)-n = n-2k$. Furthermore, the number of replicas in this intersection that store their states on non-volatile storage is $\geq (n - 2k) + (2k + r + 1) - n = r + 1$. Since at most r replicas experience rollback failures, there exists at least one replica p in the intersection of the quorums used by WQ and RQ that stores its state on non-volatile storage and never rolls back. Our protocol only allows replicas to increase the value of state.ts during normal execution (line 22), so replica p must have state.ts $\geq x$ after it responds to WQ. Since RQ starts after WQ completes, p must respond to RQ after WQ completes. Thus, p must have state.ts $\geq x$ when it responds to RQ, so RQ receives a response containing state.ts $\geq x$. ◀

Algorithm $\mathcal{A}_2$. Algorithm $\mathcal{A}_2$ handles the case when the predicate $P(n, k, r, b)$ does not hold (the right and top-left boxes in Figure 1 and the second disjunct in Theorem 2). The algorithm does not use non-volatile storage, and has write quorums of size $q_w = n - k$ and read quorums of size $q_r = k + 1$. Additionally, only replicas that have not crashed before being queried can participate in read quorums. To ensure this, each replica sets its stale flag to TRUE before returning from the recovery procedure (line 7). Recall that a replica contacted by `read_quorum` checks this flag and responds only if it is FALSE (line 36).

When $n \geq 2k + b + 1$, in every execution at least $n - k > 0$ replicas are eventually up, so some write quorum is eventually available. Also, at least $n - (k + b) \geq k + 1$ replicas never crash, so some read quorum is always available. Hence, in this case $\mathcal{A}_2$ is wait-free. Furthermore, $\mathcal{A}_2$ is always safe, even when $n < 2k + r + 1$. Similarly to $\mathcal{A}_1$, the safety of $\mathcal{A}_2$ this follows from the Real-Time Order Property, proved below; the rest of the proof is given in [26, §A].

Proof of Property 3 for $\mathcal{A}_2$. The intersection of any read and write quorums has a cardinality $\geq (n - k) + (k + 1) - n \geq 1$. Hence there exists a replica p that both executes TSVal$(x, _)$ and responds to RQ. Since p is a member of a read quorum, it could not have crashed. Since the value of state.ts is non-decreasing in the absence of crashes (line 22), p must have state.ts $\geq x$ after it responds to WQ. Since RQ starts after WQ completes, p must respond to RQ after WQ completes. Thus, p must have state.ts $\geq x$ when it responds to RQ, so RQ receives a response containing state.ts $\geq x$. ◀

4 Register Implementations in Dynamic Failure Models

In this section we study the implementability of registers under failure models where any number of replicas can experience CRR failures throughout an execution. By Theorem 1, no register implementation can be simultaneously safe and live in all executions if $n < 2k + \min(b, r) + 1$. To circumvent this impossibility, we introduce a family of *dynamic* failure models CRR-D, defined as follows.

We assume a constant Δ such that, in any execution, every message sent by a process p_i to a process p_j at time t is guaranteed to be received by p_j by $t + \Delta$, provided p_i and p_j are up in $[t, t + \Delta]$. Note that this assumption does not make the model stronger than asynchronous: since we do not assume a lower bound on message delays or processing times, processes do not have a means to measure time passage and thus take advantage of the existence of Δ [6].

▶ **Definition 4.** *Given d, c, and M, an execution α is* $\mathsf{CRR\text{-}D}(d, c, M)$***-valid*** *if there exists $U \subseteq \mathcal{R}$ such that $|U| \geq n - d$, all replicas in U are eventually up in α and either:*

1. *$c > d$ and for any two crash events occurring at times t and $t' \geq t$, we have $t' - t > M\Delta$; or*
2. *no replica crashes in some set $S \subseteq U$ such that $|S| \geq n - c$.*

Intuitively, the parameter d captures the upper bound on the number of replicas that are not eventually up in α, and must be known to any register implementation. Conditions 1 and 2 further restrict failure scenarios: either failures must be separated by at least M message delays, or at most $c \geq d$ replicas can crash in α. The former is similar to the churn-limiting assumptions used to model process participation in dynamic and reconfigurable systems [6]. The parameter M is implementation-specific and captures the minimum time required for a replica to recover its state before the next crash occurs.

We show that every static failure model $\mathsf{CRR}(k, r, b)$ from §3 is a special case of a dynamic model $\mathsf{CRR\text{-}D}(k, k + b, _)$. Thus, any $\mathsf{CRR\text{-}D}(k, k + b, _)$-tolerant register implementation is also $\mathsf{CRR}(k, r, b)$-tolerant.

▶ **Proposition 5.** $\forall k, r, b, M.\ \mathsf{CRR}(k, r, b) \preceq \mathsf{CRR\text{-}D}(k, k + b, M)$.

Proof. Consider any $\mathsf{CRR}(k, r, b)$-valid execution α. Then in this execution up to k replicas are faulty and up to b replicas are benign. Let U be the set of correct replicas, and $S \subseteq U$ be the set of perfect replicas. Then $|U| \geq n - k$ and $|S| \geq n - k - b$. Furthermore, the replicas in U are eventually up and those in S never crash, so the condition in case 2 of Definition 4 is satisfied. Thus, α is $\mathsf{CRR\text{-}D}(k, k + b, M)$-valid, as needed. ◀

We prove the following upper bound in the dynamic failure model:

▶ **Theorem 6.** *For all d, c, if $n \geq d + c + 1$ and $M \geq 12$, then there exists an implementation of an atomic wait-free MWMR register that is $\mathsf{CRR\text{-}D}(d, c, M)$-live and always safe.*

For the lower bound, we prove a stronger result that holds for $\mathsf{CRR\text{-}D}(d, c, M)$-tolerant implementations, and not just those that are $\mathsf{CRR\text{-}D}(d, c, M)$-live and always safe:

▶ **Theorem 7.** *For all d, c, M, if there exists a $\mathsf{CRR\text{-}D}(d, c, M)$-tolerant implementation of a safe obstruction-free SWSR register, then $n \geq d + c + 1$.*

The two theorems imply a tight resilience bound:

▶ **Theorem 8.** *For all d, c, $\mathsf{CRR\text{-}D}(d, c, M)$-live implementation of an atomic wait-free MWMR register exists for some M if and only if $n \geq d + c + 1$.*

The lower bound in Theorem 7 reveals an inherent trade-off: while by Proposition 5 any $\mathsf{CRR\text{-}D}(k, k + b, _)$-tolerant register implementation is also $\mathsf{CRR}(k, r, b)$-tolerant, any such implementation requires $n \geq 2k + b + 1$, thus sacrificing the optimal resilience under $\mathsf{CRR}(k, r, b)$ (cf. Theorem 2).

In the rest of this section we present an algorithm that validates Theorem 6, which we call TEE-Rex; we defer the proofs of the other results to [26, §B]. The algorithm's latency in crash-free executions is 4 message delays for both reads and writes, which is optimal. Since under $\mathsf{CRR\text{-}D}(d, c, M)$, any number of replicas can suffer rollbacks in some executions, the algorithm does not rely on non-volatile storage. It also does not require the knowledge of c.

4.1 Crash-Consistency Basics

```
Constants:
1  (q_w, q_r) ≜ (n − d, d + 1)

State:
2  state, initially ((0,0), v_0, [0..0], [0..0]), with selectors ts, val, cv, and pre_cv
3  stale ∈ {TRUE, FALSE}, initially FALSE

4  on restart
5      stale ← TRUE
6      S ← read_quorum(preCV(i))
7      next_inc ← max(S) + 1
8      write_quorum(preCV(i, next_inc))
9      state.cv[i] ← next_inc
10     write_quorum(CV(i, state.cv[i]))
11     S ← read_quorum(State(state.cv[i]))
12     forall z = 1..n do
13         state.pre_cv[z] ← max{state.pre_cv[z],
             max_{s∈S}{s.pre_cv[z]}}
14         state.cv[z] ← max{state.cv[z],
             max_{s∈S}{s.cv[z]}}
15     let (ts, v) be such that ∃s ∈ S.
         (ts, v) = (s.ts, s.val) ∧ ts = max{s.ts | s ∈ S}
16     if ts > state.ts then
17         (state.ts, state.val) ← (ts, v)
18     stale ← FALSE

19 function write_quorum(req)
20     (id, Q, cv) ← (get_unique_id(), ∅, [0..0])
21     do
22         do
23             foreach p_z ∈ R \ Q do
24                 send WRITE(id, req, cv[z]) to p_z
25         periodically until received
             {WRITE_ACK(id, inc_j, cv_j, j) | p_j ∈ K}
           for some K such that |K ∪ Q| ≥ q_w
26         if req ∈ {TSVal(_, _)} then
27             S ← {cv_j | p_j ∈ Q ∪ K}
28         else  %req ∈ {CV(_, _), preCV(_, _)}
29             S ← read_quorum(CV)
30         forall z = 1..n do
31             cv[z] ← max{cv'[z] | cv' ∈ S}
32         Q ← {p_j ∈ Q ∪ K | inc_j ≥ cv[j] ∨ i = j}
33     until |Q| ≥ q_w

34 function read_quorum(req)
35     id ← get_unique_id()
36     do
37         send READ(id, req) to R
38     periodically until received
         {READ_ACK(id, x_j, j) | p_j ∈ Q}
       for some Q such that |Q| ≥ q_r
39     return {x_j | p_j ∈ Q}

40 when received READ(id, req) from p_j
41     pre: stale = FALSE
42     if req = State(inc) then
43         state.cv[j] ← max{state.cv[j], inc}
44     r ← case req do
45             TS: state.ts
46             TSVal: (state.ts, state.val)
47             preCV(j): state.pre_cv[j]
48             CV: state.cv
49             State(inc): state
50     send READ_ACK(id, r, i) to p_j

51 when received WRITE(id, TSVal(ts, v), _)
     from p_j
52     pre: stale = FALSE
53     if ts > state.ts then
54         (state.ts, state.val) ← (ts, v)
55     send WRITE_ACK(id, state.cv[i], state.cv, i)
       to p_j

56 when received WRITE(id, req, inc) from p_j
57     pre: req ∈ {CV(j, v), preCV(j, v)}
58     state.cv[i] ← max{state.cv[i], inc}
59     if req = preCV(j, v) then
60         state.pre_cv[j] ←
             max{state.pre_cv[j], v}
61     else
62         state.cv[j] ← max{state.cv[j], v}
63     send WRITE_ACK(id, state.cv[i], ⊥, i) to p_j
```

Figure 3 Pseudocode of TEE-Rex at process $p_i \in \mathcal{P}$.

The pseudocode of TEE-Rex appears in Figure 3. It reuses the `read` and `write` procedures of algorithm $\mathcal{A}$ (Figure 2, lines 8–28). The algorithm relies on read quorums of size $q_r = d + 1$, and so-called *crash-consistent* write quorums of size $q_w = n - d$, introduced in the following. Given the mapping between static and dynamic models established by Proposition 5, these quorum sizes mirror those in $\mathcal{A}_2$: $k + 1$ for read quorums and $n - k$ for write quorums.

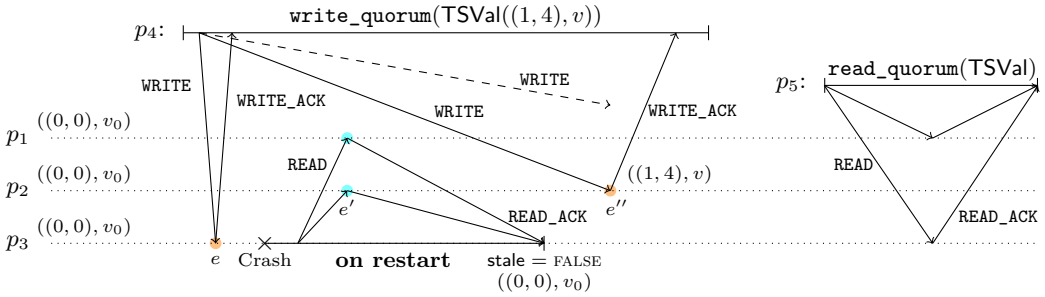


Figure 4 An execution with incorrect recovery.

Since replicas lose their memory contents during a restart, TEE-Rex follows the approach of $\mathcal{A}_2$: a replica relies on its stale flag to determine whether its current state can be used to respond to READ requests. However, in contrast to $\mathcal{A}_2$, simply setting stale = TRUE upon restart so that any replica that has not crashed permanently is active will not be live. To see why, fix arbitrary M and Δ, and consider an execution α where all replicas are eventually up, every replica crashes and restarts at least once, and every two consecutive crashes are separated by $M\Delta$. Clearly, α is CRR-D$(0, n, M)$-valid. However, $\mathcal{A}_2$ will not be live in α as it eventually runs out of non-stale replicas to form read quorums. To deal with such scenarios, a key ingredient of TEE-Rex is a novel recovery protocol executed by a replica upon restart (lines 4–18). This protocol reconstructs the state of a restarted replica, and thus enables it to clear its stale flag.

The protocol achieves this by synchronizing with other replicas. However, doing this naively by simply querying a read quorum and adopting the state of the most up-to-date replica would lead to a safety violation. For example, consider the algorithm $\mathcal{A}_2$ instantiated with $q_r = d + 1$ and $q_w = n - d$. Suppose we modify the **on restart** procedure at line 6 as follows. After setting stale = TRUE, a restarting replica would invoke read_quorum(TSVal) to retrieve the register states from a read quorum of replicas, assign the one with the highest timestamp to its own copy of state, set stale = FALSE, and then return. The following scenario demonstrates that this modification results in a linearizability violation.

▶ **Example 9.** Let $n = 3$, $d = 1$ and $c = 1$, so that $n \geq d + c + 1$. Consider the execution of the modified algorithm in Figure 4, which is valid under CRR-D$(1, 1, _)$. First, a client p_4 invokes write(v) (line 8 in Figure 2), which calls read_quorum(TS) to query timestamps at a read quorum and then write_quorum(TSVal$((1, 4), v)$) to store the new register state at a write quorum. In the latter call, p_4 sends $m =$ WRITE(TSVal$((1, 4), v)$) to all replicas and waits until it gathers WRITE_ACK responses from a write quorum. Replica p_3 gets m first, sets state $= ((1, 4), v)$, and responds with a WRITE_ACK. It then crashes, loses its state, and reconstructs the state by contacting a read quorum $\{p_1, p_2\}$ of size $q_r = d + 1 = 2$. Since m has not yet reached either replica, p_3 sets state back to its initial value $((0, 0), v_0)$. Replica p_2 then receives m, sets state $= ((1, 4), v)$, and replies to p_4 with a WRITE_ACK. Process p_4 now gathers WRITE_ACK responses from $q_w = n - d = 2$ processes, which constitutes a write quorum. Thus, write_quorum and write(v) at p_4 terminate. Later, a client p_5 calls read, which obtains responses from a read quorum $\{p_1, p_3\}$. Since neither replica is aware of p_4's write, the read returns v_0, violating linearizability.

To address the above problem, the set of responses collected by p_4 from a write quorum of replicas must be *crash-consistent* [36]. Let $\rightarrow$ be a partial order on the set of replica events such that $e \rightarrow e'$ if there exists a replica p_i such that e precedes e' at p_i.

▶ **Definition 10.** *A set E of* [**send** `WRITE_ACK`] *events is* ***crash-consistent*** *if for for each event $e \in E$ occurring at replica p_i, if after this event p_i crashes, restarts, and invokes* `read_quorum`(State)*, then the set E' of* [**send** `READ_ACK`] *events triggered by the read satisfies $\neg\exists e' \in E'.\ \exists e'' \in E.\ e' \rightarrow e''$.*

For example, consider Figure 4. Let E be the set of [**send** `WRITE_ACK`] events triggered by the call to $wq =$ `write_quorum`(TSVal$((1,4),v)$) at p_4 (orange dots), and let E' be the set of [**send** `READ_ACK`] events triggered by the call to `read_quorum`(State) during the recovery at p_3 (blue dots). Let $e \in E$ be the event at p_3 sending `WRITE_ACK`, $e' \in E'$ the event at p_2 sending `READ_ACK`, and $e'' \in E$ the event at p_2 sending `WRITE_ACK`. Since $e' \rightarrow e''$, the set E is not crash-consistent.

4.2 Crash-Consistency with Incarnation Numbers and Crash Vectors

To implement crash-consistency, similarly to [25, 36], each replica in TEE-Rex maintains two pieces of metadata – an *incarnation number* and a *crash vector*. The former is an integer, initialized to 0, and the latter tracks the highest known incarnation numbers of all other replicas. The crash vector of replica p_i is stored in state.cv, and for convenience, p_i's own incarnation number is stored in state.cv$[i]$. This incarnation number is updated by the recovery procedure at line 9. To explain the basic principles underlying crash-consistency checks in TEE-Rex, we first assume that incarnation numbers written by each replica p_i into state.cv$[i]$ monotonically increase:

▶ **Property 11** (Incarnation Number Monotonicity)**.** *Assume that a replica p_i sets* state.cv$[i]$ *to g at line 9 at time t. Then $g > 0$ and for all times $t' > t$, if p_i sets* state.cv$[i]$ *to g' at line 9 at time t', then $g' > g$.*

This property can be easily ensured assuming that each replica has a built-in monotonic counter that is *never* rolled back, as done in previous work [36]. In §4.3 we show how to ensure this property without such an assumption. The code needed for this is shown in blue and should be ignored when reading this section.

`write_quorum`(TSVal) implementation. In TEE-Rex, a process receiving a `WRITE` message piggybacks its incarnation number and a copy of its crash vector on the `WRITE_ACK` response (line 55). These are used by the implementation of `write_quorum`(TSVal) to check whether the set of `WRITE_ACK` responses it receives is crash-consistent.

This check is integrated within the loop executed by `write_quorum` (line 21). At each iteration of this loop, a `WRITE` message is (periodically) broadcast to all replicas excluding those in the set Q, which accumulates the replicas whose responses have been already validated as crash-consistent. Once acknowledgments are received from a set K of replicas such that $K \cup Q$ is a write quorum (line 25), the function checks their crash-consistency. The replicas that sent responses that are not crash-consistent are then purged from $K \cup Q$, and the set of remaining replicas is reassigned to Q (line 32).

To identify which replicas p_j in the set $K \cup Q$ sent crash-consistent responses, the crash vectors received in the `WRITE_ACK` messages are combined into a vector cv by taking their entry-wise maximum (line 31). Then the incarnation numbers inc_j in the `WRITE_ACK` messages are compared against $cv[j]$. If $inc_j < cv[j]$, then p_j restarted while our invocation of `write_quorum` was gathering `WRITE_ACK` responses from other replicas, which indicates that p_j's prior response is no longer crash-consistent. As a special case, a replica always treats its own `WRITE_ACK` responses as crash consistent (the $i = j$ disjunct at line 32). If after excluding all replicas whose responses are not crash-consistent, Q contains a write quorum, then `write_quorum` returns (line 33); otherwise, it proceeds to the next iteration of the loop.

Recovery implementation. The recovery procedure executed by replica p_i starts by setting its stale flag to TRUE (line 5), thus preventing p_i from replying to `READ` and `WRITE`(TSVal) messages (lines 41 and 52). The replica then selects a new incarnation number (line 9), which is required to satisfy Property 11, and stores it at a write quorum (line 10; we explain the implementation of `write_quorum`(CV) later). The replica further invokes `read_quorum`(State) to retrieve the states from a read quorum, including both register states and crash vectors (line 11). Finally, the replica reconstructs its state by merging the crash vectors it received and picking the register value with the highest timestamp, and clears the stale flag (lines 12–18).

The `READ`(State) messages sent by the `read_quorum` to retrieve the states (line 11) carry the new incarnation number, and a replica receiving such a message incorporates the new incarnation number into its crash vector (line 43). Thus, the new incarnation number is written to a read quorum of replicas, which intersects any write quorum that may be used by a concurrent invocation of `write_quorum`(TSVal). This ensures that this invocation only accepts a crash-consistent set of responses (Definition 10) and rules out the execution in Example 9, as we now explain.

▶ **Example 12.** Assume that in Figure 4, p_3's incarnation number before the crash is g_1 and its new incarnation number after the recovery is $g_2 > g_1$. Then p_3 will send g_1 in its `WRITE_ACK` response to p_4 (line 55), and g_2 in its `READ`(State(g_2)) request to p_2 (line 11). When p_2 responds to p_3, it will record g_2 in its crash-vector entry for p_3 (line 43). Then p_2 will piggyback this crash vector on its `WRITE_ACK` response to p_4's `WRITE` request (line 55). This will cause p_4 to discard p_3's `WRITE_ACK` response, since it carries a smaller incarnation number (g_1) than p_3's entry in the crash vector received from p_2 (g_2) (line 32).

Note that in the above example, replica p_2 might crash and recover after responding to p_3 but before responding to p_4. When recovering, p_2 will reconstruct its crash vector to a value no lower than what it was before the crash (line 14), thus giving a correct response to p_4. This is ensured by the fact that each recovering replica, such as p_3, writes its incarnation number crash-consistently to a write quorum (line 10), which intersects with a read quorum that a replica such as p_2 uses to reconstruct its state during recovery (line 11).

Thus, a replica p_i writes its incarnation number in the restart handler in two places – first crash-consistently to *some write quorum* (line 10), and then to the *particular read quorum* used to reconstruct its state (line 11). These serve complementary purposes: the former ensures that other replicas can reconstruct the incarnation number of p_i when they restart; the latter ensures that other replicas can detect when the crash-consistency of their writes can be compromised by a concurrent restart of p_i.

`write_quorum`(CV) implementation. We now describe the `write_quorum` implementation for incarnation numbers (line 10), which happens to be subtle. We could implement `write_quorum`(CV) in the exact same way as `write_quorum`(TSVal), by using crash vectors piggybacked on `WRITE_ACK` messages to check for crash-consistency of a write; this was the approach taken in [36]. Unfortunately, the resulting algorithm would not be live cases where liveness must be ensured to match the lower bound of Theorem 7.

▶ **Example 13.** Let $n = 5$, $d = 1$ and $c = 3$, so that $n \geq d + c + 1$. Consider an execution of the above version of the algorithm where p_1 crashes permanently, p_2 and p_3 crash and restart once at the beginning of the execution, and the remaining replicas never crash. This execution is valid under CRR-D$(1, 3, _)$ (case 2 in Definition 4). But p_2 and p_3 would not be able to complete the recovery in it, because their calls to `write_quorum`(CV) would wait for $q_w = n - d = 4$ responses (line 25). However, only active replicas would respond to `WRITE` messages (line 52), and there are only 2 such replicas.

To rule out such executions, in our algorithm a replica accepts a `WRITE` message for a write to state.cv even if its stale flag is set (line 56). In this case the replica cannot piggyback its crash vector on the `WRITE_ACK` response (as in line 55): the replica may have lost the crash vector upon a restart and has not yet reconstructed it (line 14). Hence, the replica puts a dummy value $\perp$ in place of a crash vector (line 63). The `write_quorum` function then checks crash-consistency for writes of incarnation numbers differently from register writes: after it receives enough `WRITE_ACK` responses, it calls `read_quorum`(CV) to read the crash vectors explicitly (line 29). Replicas only reply to `READ` requests sent by this function when their stale flag is cleared (line 41), and thus they have valid crash vectors in their state.

In the absence of further crashes in Example 13, p_2 and p_3 can complete the recovery as follows: they first collect `WRITE_ACK`s from $q_w = n - d = 4$ replicas that are up (p_2, p_3, p_4, p_5) and then read crash vectors from $q_r = d + 1 = 2$ replicas that are active (p_4, p_5). We can show that TEE-Rex is correct assuming Incarnation Number Monotonicity.

▶ **Theorem 14.** *If incarnation numbers assigned at line 9 satisfy Property 11, then the algorithm in Figure 3 excluding the highlighted lines satisfies the conditions of Theorem 6.*

We defer the proof of the theorem to [26, §B]. As in §3.1, we show the safety of the algorithm by first establishing the Real-Time Property (Property 3). We next explain how to ensure Incarnation Number Monotonicity.

4.3 Implementing Incarnation Numbers without Non-Volatile Storage

Since in the presence of rollbacks the replicas cannot rely on non-volatile storage to implement monotonically growing incarnation numbers, TEE-Rex relies on a distributed mechanism for this purpose, shown in blue in Figure 3. This mechanism extends the one used to store incarnation numbers in state.cv (line 10) described in the previous section. Note that a replica cannot restore its previous incarnation number by just reading it using `read_quorum`(CV): if the replica crashed before completing the `write_quorum`(CV) at line 10, the `read_quorum` is not guaranteed to restore its latest pre-crash incarnation number, thus violating Property 11.

To address this problem, we adopt a two-phase approach. In addition to cv, the state of each replica includes a vector pre_cv. Before assigning its new incarnation number to state.cv$[i]$ (line 9), a replica p_i first writes it to state.pre_cv$[i]$ at a crash-consistent write quorum of replicas (line 8). Upon a restart, a replica p_i restores its previous incarnation number as a maximum of responses returned by `read_quorum`(preCV(i)) and computes the new incarnation number by incrementing the result (lines 6–7). Then the intersection between the quorums used by `write_quorum`(preCV) (line 8) and `read_quorum`(preCV) (line 6) helps ensure Property 11.

There is, however, a subtlety. Recall that in our algorithm a replica has to reply to a `WRITE` message for a write to state.cv even while it restarting (line 56), and the same should hold for writes to state.pre_cv: this is necessary to avoid deadlocks like the one in Example 13. When incarnation numbers are computed as described above, a restarting replica thus needs to reply to a `WRITE` even before it computed its incarnation number at line 9. This poses a dilemma: which incarnation number should the replica include into its `WRITE_ACK` message (line 55) to allow other replicas to validate the crash-consistency of their writes to state.cv and state.pre_cv? To address this challenge, when a replica p_j executing `write_quorum`(CV) or `write_quorum`(preCV) sends a `WRITE` message to another replica p_i, it piggybacks the maximum incarnation number of p_i known to it on this message (line 24). A replica p_i receiving a `WRITE` request for state.pre_cv$[i]$ or state.cv$[i]$ then adopts this incarnation number (line 58) and uses it in its `WRITE_ACK` reply.

In [26, §B] we prove that, even though the above mechanism may require processes to *temporarily* adopt stale incarnation numbers, it does ensure Incarnation Number Monotonicity (Property 11). Thus, given Theorem 14, the overall TEE-Rex algorithm is correct. Here we illustrate the operation of the incarnation number implementation on an example.

▶ **Example 15.** Let $n = 5$, $d = 1$ and $c = 3$, so that $n \geq d + c + 1$. Consider an execution where p_1 crashes permanently, p_2 and p_3 are eventually up, and the remaining replicas never crash. This execution is valid under CRR-D$(1, 3, _)$ (case 2 in Definition 4). At the beginning of the execution, p_2 and p_3 crash, restart, and become active again, so that p_4 and p_5 have state.cv$[2]$ = state.cv$[3] = 1$. Suppose now that p_2 and p_3 crash and restart again.

To recover, each of p_2 and p_3 determines its previous incarnation number 1 (line 6) and computes a new incarnation number 2, which it tries to store crash-consistently in state.pre_cv at a write quorum (line 8). Since p_2 does not have any information about the incarnation number of p_3, its first `WRITE` message to p_3 carries 0 as the incarnation number (line 24), which p_3 includes into its `WRITE_ACK` reply. The `read_quorum`(CV) that p_2 invokes after this (line 29) fetches 1 for state.cv$[3]$ from a read quorum $\{p_4, p_5\}$, so the `WRITE_ACK` from p_3 is discarded (line 32). The same happens at p_2 with the `WRITE_ACK` it receives from p_3.

Each of p_2 and p_3 then includes 1 into the next `WRITE` message it sends to the other replica (line 24), which the latter adopts as its incarnation number and includes into its `WRITE_ACK` reply (line 58). Once p_2 receives the `WRITE_ACK` from p_3, it calls `read_quorum`(CV) once more (line 29) and validates the `WRITE_ACK` as crash-consistent, finishing the execution of `write_quorum`(preCV$(2, 2)$). Replica p_2 then sets state.cv$[2]$ to its new incarnation number 2 (line 9) and completes the rest of the recovery, including `write_quorum`(CV$(2, 2)$).

Assume that now p_3 receives the `WRITE_ACK` from p_2 and calls `read_quorum`(CV) to validate it (line 29). This read yields 2 for state.cv$[2]$, so p_3 discards the `WRITE_ACK` and sends another `WRITE` to p_2 with incarnation number 2 (line 24). Now p_2 responds with a `WRITE_ACK` carrying its incarnation number 2, which it also stores locally. When p_3 receives this `WRITE_ACK`, it performs another `read_quorum`(CV), validates the `WRITE_ACK` as crash-consistent and completes `write_quorum`(preCV$(3, 2)$). It then sets state.cv$[3]$ to its new incarnation number 2 (line 9) and completes the rest of the recovery. In the end, both p_2 and p_3 successfully recover with a higher incarnation number, satisfying Property 11.

5 Related Work

The classical crash-recovery failure model assumed that every process is equipped with durable storage that cannot be rolled back [22]. This assumption was lifted by Aguilera et al. [4], who analyzed consensus solvability as a function of the number of processes with and without durable storage. The same model was also used by Guerraoui et al. [17] to derive tight bounds for a reliable register construction. In these models the processes are assumed to know whether they are equipped with durable storage, which they can trust to be incorruptible. In contrast, in our CRR model a recovering process cannot trust its non-volatile storage to be up-to-date, leading to different resilience bounds. Furthermore, in the register implementation of Guerraoui et al. a restarted replica is considered up-to-date once it accepts a single `WRITE` request. As we show in [26, §C], this leads to a safety violation.

Dinis et al. [13] proposed a rollback-recovery (RR) model to capture rollback attacks in TEE-based systems. This model is weaker than CRR as it disallows permanent process crashes. Furthermore, the register implementation of Dinis et al. uses ordinary quorums for writes, without crash-consistency checks. As we show in [26, §D], this leads to a safety

violation similar to the one in Example 9. The implementation also does not guarantee wait-freedom for reads. In contrast, our CRR framework captures the full spectrum of failures in TEE-based systems and enables developing correct solutions for these environments.

The crash-recovery failure model where the processes do not have access to incorruptible durable storage was assumed by Chandra et al. [10], Kończak et al. [27], and Liskov et al. [30] in the context of their efforts to develop practical variants of Paxos [29] and viewstamped replication [38]. The proposed solutions, however, did not use crash-consistent quorums for storing their state, and as a result, were shown in [36] to violate safety.

Jehl et al. [25] proposed a versioning scheme similar to crash vectors that allows quickly replacing a failed replica in Paxos. This technique was subsequently generalized to the notion of crash-consistent quorums by Michael et al. [36], who also demonstrated how it can be used to implement a recoverable atomic register. However, the safety of these versioning schemes critically depends on the replicas' ability to track their incarnations across restarts. In turn, supporting this capability in TEE-based systems requires hardware-based persistent counters. These counters are implemented using flash memory, resulting in poor write performance and quick wear-out [35]. Furthermore, they are not universally supported by TEE manufacturers, and have been recently deprecated by Intel SGX [23].

Although the technical report version [37] of the work by Michael et al. suggests that such incarnation numbers can be supported by means of a distributed mechanism, it does not provide a full implementation. In addition, while the recoverable register implementation of [36, 37] is always safe, its liveness under the static failure models requires $n = 2c + 1$, which is strictly worse than our bound of $n \geq d + c + 1$.

Persistent monotonic hardware counters have been demonstrated in [40] to be a powerful mechanism to guard against rollback attacks. In TEE-based systems they can be used together with sealing (§1) to validate state freshness upon restart [12]. Unfortunately, these solutions inherit the drawbacks of persistent counters we explained above.

ROTE [35] guards against rollbacks using a rollback-resistant distributed counter. Its implementation relies on a two-round protocol where a new value of the counter is first written to a write quorum, and then read back from the *same* write quorum to validate that it was stored reliably. Although this solution is safe, it does not ensure liveness, as the same write quorum cannot be guaranteed to be available two times in a row under failures and asynchrony. Engraft [42] uses a similar protocol, so is subject to the same liveness issue.

References

1 Ittai Abraham, Gregory V. Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk Paxos: optimal resilience with Byzantine shared memory. *Distributed Comput.*, 18(5), 2006. doi:10.1007/S00446-005-0151-6.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of Byzantine broadcast: A complete categorization. In *Symposium on Principles of Distributed Computing (PODC)*, 2021.

3 Yehuda Afek, David S. Greenberg, Michael Merritt, and Gadi Taubenfeld. Computing with faulty shared objects. *J. ACM*, 42(6), 1995. doi:10.1145/227683.227688.

4 Marcos K Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Comput.*, 13(2), 2000. doi:10.1007/S004460050070.

5 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. In *Symposium Principles of Distributed Computing (PODC)*, 1990.

6 Hagit Attiya, Hyun Chul Chung, Faith Ellen, Saptaparni Kumar, and Jennifer L. Welch. Simulating a shared register in an asynchronous system that never stops changing. In *Symposium on Distributed Computing (DISC)*, 2015.

7 Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: SGX-based high performance BFT. In *European Conference on Computer Systems (EuroSys)*, 2017.

8 Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and forking detection for trusted execution environments using lightweight collective memory. In *Conference on Dependable Systems and Networks (DSN)*, 2017.

9 Dhruva R. Chakrabarti, Hans-Juergen Boehm, and Kumud Bhandari. Atlas: leveraging locks for non-volatile memory consistency. In *Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2014.

10 Tushar D Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: An engineering perspective. In *Symposium on Principles of Distributed Computing (PODC)*, 2007.

11 Gregory Chockler, Nancy Lynch, Sayan Mitra, and Joshua Tauber. Proving atomicity: An assertional approach. In *Symposium on Distributed Computing (DISC)*, 2005.

12 Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016. URL: `http://eprint.iacr.org/2016/086`.

13 Baltasar Dinis, Peter Druschel, and Rodrigo Rodrigues. RR: A fault model for efficient TEE replication. In *Network and Distributed System Security Symposium (NDSS)*, 2023.

14 Dan Dobre, Rachid Guerraoui, Matthias Majuntke, Neeraj Suri, and Marko Vukolic. The complexity of robust atomic storage. In *Symposium on Principles of Distributed Computing (PODC)*, 2011.

15 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *J. ACM*, 32(1), 1985. `doi:10.1145/2455.214112`.

16 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2), 1988. `doi:10.1145/42282.42283`.

17 Rachid Guerraoui, Ron R. Levy, Bastian Pochon, and Jim Pugh. The collective memory of amnesic processes. *ACM Trans. Algorithms*, 4(1), 2008. `doi:10.1145/1328911.1328923`.

18 Rachid Guerraoui and Marko Vukolic. How fast can a very robust read be? In *Symposium on Principles of Distributed Computing (PODC)*, 2006.

19 Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1), 1991. `doi:10.1145/114005.102808`.

20 Maurice Herlihy, Victor Luchangco, and Mark Moir. Obstruction-free synchronization: Double-ended queues as an example. In *International Conference on Distributed Computing Systems (ICDCS)*, 2003.

21 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3), 1990. `doi:10.1145/78969.78972`.

22 Michel Hurfin, Achour Mostéfaoui, and Michel Raynal. Consensus in asynchronous systems where processes can crash and recover. In *Symposium on Reliable Distributed Systems (SRDS)*, 1998.

23 Intel Corporation. End of life for Intel SGX monotonic counter service. `https://www.intel.com/content/www/us/en/support/articles/000057968/software/intel-security-products.html`, 2023. Accessed: 2025-02-09.

24 Prasad Jayanti, Tushar Deepak Chandra, and Sam Toueg. Fault-tolerant wait-free shared objects. *J. ACM*, 45(3), 1998. `doi:10.1145/278298.278305`.

25 Leander Jehl, Tormod Erevik Lea, and Hein Meling. Replacement: Decentralized failure handling for replicated state machines. In *Symposium on Reliable Distributed Systems (SRDS)*, 2015.

26 Sadegh Keshavarzi, Gregory Chockler, and Alexey Gotsman. TEE is not a healer: Rollback-resistant reliable storage (extended version). *arXiv*, 2505.18648, 2025. URL: `https://arxiv.org/abs/2505.18648`.

27 Jan Kończak, Nuno Santos, Tomasz Żurkowski, Paweł T. Wojciechowski, and André Schiper. JPaxos: State machine replication based on the Paxos protocol. Technical Report EPFL-REPORT-167765, EPFL, 2011.

28 Leslie Lamport. On interprocess communication. *Distributed Comput.*, 1(2), 1986.

29 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2), 1998. doi:10.1145/279227.279229.

30 Barbara Liskov and James Cowling. Viewstamped replication revisited. Technical Report MIT-CSAIL-TR-2012-021, MIT CSAIL, 2012.

31 Jian Liu, Wenting Li, Ghassan O. Karame, and N. Asokan. Scalable Byzantine consensus via hardware-assisted secret sharing. *IEEE Trans. Computers*, 68(1), 2019. doi:10.1109/TC.2018.2860009.

32 Nancy A Lynch and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Comput.*, 1(1), 1986. doi:10.1007/BF01843568.

33 Nancy A. Lynch and Alexander A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Symposium on Distributed Computing (DISC)*, 2002.

34 Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Minimal Byzantine storage. In *Symposium on Distributed Computing (DISC)*, 2002.

35 Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback protection for trusted execution. In *USENIX Security Symposium (USENIX Security)*, 2017.

36 Ellis Michael, Dan R. K. Ports, Naveen Kr. Sharma, and Adriana Szekeres. Recovering Shared Objects Without Stable Storage. In *Symposium on Distributed Computing (DISC)*, 2017.

37 Ellis Michael, Dan R. K. Ports, Naveen Kr. Sharma, and Adriana Szekeres. Recovering shared objects without stable storage (extended version). Technical Report UW-CSE-TR-17-10-01, University of Washington, 2017. URL: https://www.microsoft.com/en-us/research/publication/recovering-shared-objects-without-stable-storage-extended-version/.

38 Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. *Symposium on Principles of Distributed Computing (PODC)*, 1988.

39 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2), 1980. doi:10.1145/322186.322188.

40 Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In *USENIX Security Symposium (USENIX Security)*, 2016.

41 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient Byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1), 2013. doi:10.1109/TC.2011.221.

42 Weili Wang, Sen Deng, Jianyu Niu, Michael K. Reiter, and Yinqian Zhang. ENGRAFT: Enclave-guarded Raft on Byzantine faulty nodes. In *Conference on Computer and Communications Security (CCS)*, 2022.

43 Qianyu Yu, Giuliano Losa, and Xuechao Wang. TetraBFT: Reducing latency of unauthenticated, responsive BFT consensus. In *Symposium on Principles of Distributed Computing (PODC)*, 2024.

On the Randomized Locality of Matching Problems in Regular Graphs

Seri Khoury
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Manish Purohit
Google Research, Mountain View, CA, USA

Aaron Schild
Google Research, San Francisco, CA, USA

Joshua R. Wang
Google Research, San Francisco, CA, USA

Abstract

The main goal in distributed symmetry-breaking is to understand the locality of problems: the radius of the neighborhood that a node must explore to determine its part of a global solution. In this work, we study the locality of matching problems in the family of regular graphs, which is one of the main benchmarks for establishing lower bounds on the locality of symmetry-breaking problems, as well as for obtaining classification results. Our main results are summarized as follows:

1. **Approximate matching:** We develop randomized algorithms to show that $(1+\epsilon)$-approximate matching in regular graphs is truly local, i.e., the locality depends only on ϵ and is independent of all other graph parameters. Furthermore, as long as the degree Δ is not very small (namely, as long as $\Delta \geq \text{poly}(1/\epsilon)$), this dependence is only logarithmic in $1/\epsilon$. This stands in sharp contrast to maximal matching in regular graphs which requires some dependence on the number of nodes n or the degree Δ.
2. **Maximal matching:** Our techniques further allow us to establish a strong separation between the node-averaged complexity and worst-case complexity of maximal matching in regular graphs, by showing that the former is only $O(1)$.

Central to our main technical contribution is a novel martingale-based analysis for the ≈ 40-year-old algorithm by Luby. In particular, our analysis shows that applying one round of Luby's algorithm on the line graph of a Δ-regular graph results in an almost $\Delta/2$-regular graph.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Probabilistic algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases regular graphs, maximum matching, augmenting paths, distributed algorithms, Luby's algorithm, martingales

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.40

1 Introduction

Matching problems, such as maximal or maximum matching, have garnered significant attention across several computational models, including the classical sequential model [57, 64, 77, 88, 89]; dynamic networks [10, 33, 34, 59, 62, 85]; streaming algorithms [12, 13, 41, 50, 78, 82]; online algorithms [40, 61, 67, 68, 79]; and distributed computing [1, 2, 27, 32, 38, 45, 46, 48, 55, 56, 63, 66, 74, 75].

In the classical LOCAL model of distributed computing, a network of n nodes collaborates to solve a graph problem through a series of synchronized communication rounds. In each round, a node can send an unbounded-size message to each of its neighbors. The primary goal is to minimize the number of communication rounds required to solve the problem; a measure known as the problem's *locality*. Since each node can only interact with the


© Seri Khoury, Manish Purohit, Aaron Schild, and Joshua R. Wang;
licensed under Creative Commons License CC-BY 4.0



nodes in its r-hop neighborhood in r rounds, the problem's locality reflects the radius of the neighborhood that each node must explore to determine its part in the global solution (e.g., whether it is matched and if so to which neighbor).

Typically, the locality of matching problems depends on global graph parameters, such as the number of nodes n or the maximum degree Δ. For instance, a simple algorithm by Israeli and Itai [65] finds a maximal matching in $O(\log n)$ rounds in the LOCAL model, which constitutes a 2-approximate maximum matching in unweighted graphs (or approximate matching, for short). Later, Barenboim et al. [29] showed that this logarithmic dependence on n can be substituted with a logarithmic dependence on Δ by providing an $O(\log \Delta + \text{poly} \log \log n)$-round algorithm. This upper bound also applies to finding a $(1+\epsilon)$-approximate matching (while incurring a multiplicative $\text{poly}(1/\epsilon)$ factor in the number of rounds), as shown by Harris [63].

All of the above results are for randomized algorithms that succeed with high probability.[1] The landscape changes for other computational assumptions. For instance, if the matching is only required to be large in expectation, then recent work has shown that it is possible to shave a $\log \log \Delta$ factor from the round complexity [27, 28, 50, 52]. For deterministic algorithms, however, the primary question is whether the state-of-the-art randomized bounds can be matched.[2] For a more detailed discussion on deterministic algorithms, we refer the reader to the excellent surveys by Suomela [87] and Rozhon [84].

Whether there exist $o(\log n)$-round algorithms that succeed with high probability for maximal or $(1+\epsilon)$-approximate matching in general graphs remains one of the main open questions in the field. On the other hand, the best known lower bound as a function of n is $\Omega(\sqrt{\log n / \log \log n})$ rounds, which was shown by Kuhn, et al. [72], and it applies all the way up to $\text{poly}(\log n)$-approximation.

Regular Graphs. A graph is Δ-regular if all its nodes have degree Δ. The family of regular graphs has received much attention as a natural benchmark for studying the complexity of various fundamental problems (e.g., [4–9, 14–16, 42, 58, 70, 86, 90]). In the LOCAL model, regular graphs serve as the standard benchmark for implementing the *Round Elimination* technique for proving lower bounds [17,21–25,35,36,39], and also for obtaining classifications of locality results (e.g., in paths and cycles [18,37], grids [37,43,60,81], and regular trees [19,20]).[3]

The locality of some problems in regular graphs depends on graph parameters such as the number of nodes n or the degree Δ, while for others, it is independent of any such parameter. Interestingly, as we discuss next, maximal matching falls into the former category, whereas $(1+\epsilon)$-approximate matching is in the latter.

For maximal matching, the approximately 40-year-old $O(\log n)$ upper bound by Israeli and Itai [65] remains the best known even for regular graphs. Moreover, Balliu et al. [21] showed via the Round Elimination technique that maximal matching in regular graphs requires $\Omega(\min\{\Delta, \log \log n / \log \log \log n\})$ rounds for randomized algorithms and $\Omega(\min\{\Delta, \log n / \log \log n\})$ rounds for deterministic ones. Even for low-degree regular graphs (e.g. cycles), any randomized algorithm for maximal matching still requires $\Omega(\log^* n)$ rounds [73, 80].

---

[1] Throughout the paper, we say that an algorithm succeeds with high probability if it succeeds with probability $1 - 1/n^c$ for some arbitrarily large constant $c > 0$.

[2] The current state-of-the-art deterministic algorithms take $\tilde{O}(\log^{5/3} n)$ rounds for maximal matching [54], and $\tilde{O}(\log^{4/3} n \cdot \text{poly}(1/\epsilon))$ rounds for $(1+\epsilon)$-approximate matching [50, 53], where $\tilde{O}(x)$ hides $\text{poly} \log x$ factors.

[3] For more on classification of locality results, as well as ones in general graphs, we refer the reader to the survey by Rozhon [84].

In stark contrast, the story for $(1+\epsilon)$ approximate matching in regular graphs is far less clear. The hard instances for approximate matching developed by Kuhn et al. [72] are very far from being regular. In fact, these instances are also hard for finding an approximate *fractional matching*. On regular graphs, however, fractional matching admits a trivial zero-round solution by simply setting the fractional value for each edge to be $1/\Delta$. This fractional matching can be rounded to an $O(1)$-approximate integral matching in one round by using a simple sampling technique [52].

This raises the question: where does $(1+\epsilon)$-approximate matching fall on the spectrum of locality in regular graphs? Is it closer to approximate fractional matching, or does it require some dependence on n or Δ similar to maximal matching?[4]

1.1 Our Results

Approximate Matching. In this work, our first result is a simple algorithm that finds a $(1+\epsilon)$-approximate matching in regular graphs without any dependence on graph parameters.[5] For constant values of ϵ, the algorithm also works in the more restricted CONGEST model, where the size of the messages is bounded by $O(\log n)$ bits.

▶ **Theorem 1.** *Let n be a positive integer, and let $\epsilon \in (n^{-1/20}, 1/2)$ be an accuracy parameter. There is an $O(\epsilon^{-5}\log(1/\epsilon))$-round LOCAL algorithm that finds a $(1+\epsilon)$-approximate maximum matching in n-node regular graphs, with high probability. The algorithm works in the CONGEST model for constant values of ϵ.*

While Theorem 1 advances our understanding of $(1+\epsilon)$-approximate matching in regular graphs, the polynomial dependence on $1/\epsilon$ is far from desirable. Comparing with the $\tilde{O}(\epsilon^{-3}\log\Delta + \text{polylog}(1/\epsilon, \log\log n))$ round algorithm of Harris [63], the algorithm from Theorem 1 is better only in the regime $\epsilon > 1/\sqrt{\log\Delta}$. In our next result, we present an exponentially faster algorithm as long as $\epsilon > 1/\Delta^{c'}$ for some constant $c' > 0$. In other words, we present an exponentially faster $(1+\epsilon)$-approximation algorithm for graphs that are not extremely sparse, i.e., when $\Delta > \text{poly}(1/\epsilon)$. We note that the constant c in Theorem 2 has not been optimized and can be improved substantially with a more careful analysis.

▶ **Theorem 2** (Main Result I). *Let $c = 10^5$ and let $\epsilon \in (0,1)$ be an accuracy parameter. For Δ-regular graphs with $\Delta > (1/\epsilon)^c$, there is an $O(\log(1/\epsilon))$-round CONGEST algorithm that finds a $(1+\epsilon)$-approximate maximum matching with high probability.*

One may wonder whether the restriction in the theorem about the graph being dense enough is a mere technicality. The following lower bound presented in Theorem 3 shows that this is not the case, allowing us to establish a separation between dense and sparse regular graphs in which dense ones are easier. Note that, for the same problem, the *opposite* separation holds in general graphs (due to the $\Omega(\min\{\log\Delta/\log\log\Delta, \sqrt{\log n/\log\log n}\})$ lower bound by [72], and the upper bounds by [63]).

[4] Note that the amplification technique of [50] cannot be used to amplify the constant-approximation factor to $(1+\epsilon)$ in regular graphs while using $\text{poly}(1/\epsilon)$ phases of amplification. This is because the technique is designed for general graphs, and applying it to a regular graph can result in a non-regular graph after the first step. Consequently, the amplification algorithm would need to use an algorithm for constant-approximate matching in *general graphs*, which requires some dependence on n or Δ.

[5] All our upper bounds apply also to almost regular graphs, where the degrees of all the nodes are within a $(1\pm o(1))$-multiplicative factor from each other. Moreover, the case where $\epsilon \le n^{-1/20}$ can be handled in $\text{poly}(1/\epsilon)$ rounds by transmitting the entire graph to all nodes.

▶ **Theorem 3** (Informal version of hardness result; see full version for details). *For any degree $\Delta \geq 2$ and error $\epsilon = O(\Delta^{-1})$, any LOCAL algorithm that computes a $(1+\epsilon)$-approximate maximum matching in bipartite Δ-regular graphs with at least $n \geq \Omega(\Delta^{-1}\epsilon^{-1})$ nodes requires $\Omega(\Delta^{-1}\epsilon^{-1})$ rounds.*

One intuition behind this separation is that in Δ-regular graphs, the number of nearly optimal solutions to maximum matching scales with Δ (see for instance [11]). Intuitively, this abundance of nearly optimal solutions makes it easier to rapidly identify one using randomness.

At the heart of the proof of our first main result (Theorem 2) is a novel martingales-based analysis of the classical algorithm by Luby [76]. In particular, we use this analysis to prove our *Regular-Graph Preservation Lemma* (Lemma 6), which shows that running a single round of Luby's algorithm on the line graph of a sufficiently dense Δ-regular graph and removing the matched nodes together with their incident edges yields an almost $\approx \Delta/2$-regular graph. Interestingly, this analysis wilil also allow us to attain a good node-averaged complexity for maximal matching.

Node-Averaged Complexity of Maximal Matching. When performing a distributed computation, the algorithm may arrive at some parts of its final output before the rest. That is, some nodes know their local result before the algorithm fully terminates on the last few stragglers. The notion of node-averaged complexity [26, 30, 44, 47] captures this nuance by reasoning about the average over the times at which the nodes finish their computation and commit to their outputs (we formally define the node-averaged complexity in Section 1.2). Node-averaged complexity captures important applications such as optimizing the total energy consumption of the network (see, for instance, [44] and references within).

In sharp contrast to the worst-case complexity of maximal matching in regular graphs, our martingale-based analysis of Luby's algorithm yields a *constant* node-averaged complexity. This also contrasts with a lower bound of Balliu et al. [26] who showed that in general graphs, the node-averaged complexity of maximal matching is $\Omega\left(\min\left\{\frac{\log\Delta}{\log\log\Delta}, \sqrt{\frac{\log n}{\log\log n}}\right\}\right)$.[6]

▶ **Theorem 4** (Main Result II). *The node-averaged complexity of finding a maximal matching in Δ-regular graphs is $O(1)$.*

Theorem 4 implies that, with respect to node-average complexity, maximal matching is easier in regular graphs than in general ones. On the other hand, as discussed earlier, for worst-case complexity, the $O(\log n)$ bound by [65] is still the best known for both regular and general graphs. Whether maximal matching is any easier in regular graphs with respect to worst-case complexity remains an interesting open question.

Outline of the paper. In Section 1.2 we provide some basic definitions and notation. In Section 2 we provide a brief technical overview of our results. Section 3 contains a proof sketch of Theorem 2. The proof of Theorem 4 is deferred to the full version of the paper. The result is obtained by applying $O(\log\log\Delta)$ rounds of Luby, which our results show

[6] We note that the corresponding edge-averaged complexity of maximal matching is known to be $O(1)$, even in general graphs, due to the classical algorithms of [3, 76] (when applied on the line graph) that delete a constant-fraction of the edges in each round [26]. However, when an edge arrives at its part of the output (i.e., whether it is matched or not), it does not necessarily help an incident node to arrive at its part of the output, as several edges are not matched. This nuance is fundamental due to the lower bound by [26] discussed above for the node-averaged complexity of maximal matching in general graphs.

matches all but a $1/(\log \Delta)$-fraction of the nodes with $O(1)$ node-averaged complexity, and applying [51] to match the remaining nodes. The details of the proofs of Theorems 4 and 2, along with all discussion of Theorems 1 and 3, are deferred to the full version.

1.2 Model and Basic Definitions

The LOCAL and CONGEST model. In this work we are interested in the LOCAL and CONGEST models of distributed computing [73,83]. In both models, there is a synchronized communication network of n computationally unbounded nodes that can communicate via communication rounds; this network both defines the communication pattern and serves as the input graph we want to solve our problem on. In each round, each node can send an unbounded-size (in the LOCAL model) or $O(\log n)$-bit (in the CONGEST model) message to each of its neighbors. The goal is to perform a task (e.g., find a large matching) while minimizing the number of communication rounds. At the end of all communication rounds, each node needs to know which of its neighbors it is matched with, if any.

Basic Graph Notations. For a graph G, we denote by $V(G)$ the set of nodes in G and by $E(G)$ the set of edges. Given a node $u \in V(G)$, we denote by $N_G(u)$ the set of neighbors of u in G, and by $N_G^d(u)$ the set of nodes at distance exactly d from u. When G is clear from the context, we omit the letter G from the notation and use V, E and $N(u)$ for brevity. Throughout the paper, n denotes the number of nodes in the graph. In Δ-regular graphs, all the nodes have the same degree Δ. In this work, we are interested in unweighted and undirected graphs.

Maximum and maximal matching. A matching $\mathcal{M}$ in a graph G is a set of edges in $E(G)$, where no two edges in $\mathcal{M}$ share a node. A maximum matching is a matching of maximum possible size. A $(1+\epsilon)$-approximate matching in G is a matching $\mathcal{M}$ satisfying $OPT \leq (1+\epsilon)|\mathcal{M}|$, where OPT is the size of a maximum matching. A maximal matching is a matching that is not a strict subset of any other matching.

Node-averaged Complexity ([26]). The node-averaged complexity of an algorithm A, $\mathrm{AVG}_V(A)$, is defined as follows.

$$\mathrm{AVG}_V(A) := \max_{G \in \mathcal{G}} \frac{1}{|V(G)|} \cdot \sum_{v \in V(G)} \mathbb{E}\left[T_v^G(A)\right]$$

where $T_v^G(A)$ is the time it takes for v to reach its part of the output when running A. The node-average complexity of a graph problem is defined as the node-average complexity of the best algorithm A^* that minimizes $\mathrm{AVG}_V(A^*)$.

2 Technical Overview

2.1 Warmup: A $\mathrm{poly}(1/\epsilon)$-Round Algorithm for Regular Graphs

To prove Theorem 1, we use a two stage algorithm. We begin with a Δ-regular graph on n vertices with target error parameter $\epsilon > n^{-1/20}$. The first stage (sampling) involves uniformly and independently sampling edges from the graph with the goal of reducing the degree from Δ to $\mathrm{poly}(1/\epsilon)$, plus some post-processing. After this stage, we have an (irregular) graph on at most n vertices with degree at most $d = \mathrm{poly}(1/\epsilon)$ and have used up a constant fraction

of our error parameter ϵ (i.e. this restricted graph still has an almost-perfect matching). Our second stage (matching) involves finding a matching in this restricted graph, and its runtime only depends on the error parameter ϵ and the max degree d.

For the sampling stage, uniformly sampling to degree approximately $\epsilon^{-2} \log n$ would result in a graph that retains a near-perfect matching with high probability (via a Chernoff bound plus a union bound), which then when combined with Harris' algorithm [63] can already give us an $O(\text{poly}(1/\epsilon) \log \log n)$-round algorithm. However, to avoid the dependence on n, we need to find a way to reduce this degree even further. Instead, we sample down to degree $\Theta(\epsilon^{-4})$. The resulting subgraph can be very irregular, but we manage to tease out enough structure to make our argument go through. In particular, some small fraction of vertices may have degree exceeding our target $\Theta(\epsilon^{-4})$ by more than a factor two. We use Chernoff with bounded dependence and the matching polytope to argue that stripping out these problematic high-degree vertices still leaves an almost-perfect matching.

For the matching stage, we find a matching in the constructed low-degree subgraph from the sampling stage. The state-of-the-art algorithms of Harris [63] fit our task, but they have a small runtime dependence on n. Instead, we combine the hypergraph framework from [63] (which was also used in [27, 49, 50]) with some ideas from [52], as follows. In $\text{poly}(1/\epsilon)$ phases, we increase the size of the matching in each phase by $\text{poly}(\epsilon) \cdot n$ edges. The algorithm for each phase finds a large set of disjoint augmenting paths. This is done by first constructing a hypergraph H with the same set of nodes as in the low-degree subgraph, where each hyperedge corresponds to a $1/\epsilon$-length augmenting paths. Then, we find an $O(1/\epsilon)$-approximate fractional matching in the hypergraph by using the algorithms of [31,63,71], and we round this fractional matching by sampling each hyperedge with probability proportional to its fractional value. By using a similar McDiarmid-type argument as in [52], we can show that this rounding produces an integral $O(1/\epsilon)$-approximate matching in the hypergraph H with high probability. Furthermore, observe that the maximum degree of a node in the hypergraph H is $\exp(\text{poly}(1/\epsilon))$. Therefore, the algorithms of [31, 63, 71] for finding a fractional matching in this hypergraph take $O(\text{poly}(1/\epsilon))$ rounds, as desired. We discuss this result formally in the full version.

2.2 Exponentially Faster Algorithm

In this section, we give a brief technical overview for our main result. On bipartite graphs, our algorithm for proving Theorem 2 simply runs $O(\log(1/\epsilon))$ rounds of Luby's algorithm [76] that finds a large matching in each round. On general graphs, we first run a color coding step to find a large bipartite almost regular subgraph. While the algorithm is very simple, the main challenge is its analysis. In this work, we provide a new martingale-based analysis for Luby's algorithm.

Since our analysis relies on martingale concentration inequalities, it is more convenient to work with a sequential view of Luby's algorithm. Our key lemma (Lemma 6) shows that after applying one round of Luby's algorithm (and removing the matched nodes along with incident edges), the remaining graph remains almost regular, i.e., almost all nodes have very similar degrees. Even with the sequential view, classical martingale concentration inequalities are not sufficient to provide the high probability bounds that we require. We use two techniques, a *shifted martingale trick* and *scaled martingale trick*, in order to provide the requisite bounds. Roughly speaking, we show that the number of matched neighbors of a node behaves similarly to a martingale. Finally, we show that a constant fraction of the nodes are matched in each iteration of Luby's algorithm when the graph is almost regular. Combined with our key lemma, this implies that after $O(\log(1/\epsilon))$ rounds at most ϵ fraction of nodes remain unmatched. We now present a deeper overview of each of the above components in the subsections below.

2.2.1 Sequential view of Luby's algorithm

In the traditional distributed implementation of one round of Luby's algorithm, each edge f picks a uniformly random number r_f and some edge e is chosen into the matching if and only if $r_e < r_{e'}$ for all neighboring edges e'.

We consider the following sequential view - the edges of the graph arrive sequentially in a uniformly random order and an edge e is chosen into the matching if and only if it arrives before any of its neighboring edges.

Assuming that there are no collisions (i.e. each edge chooses a different random number from its neighbors), it can be readily seen that the two algorithms above produce exactly the same distribution over matchings. For CONGEST algorithms, we restrict the range of the random numbers to be integers in $\{1, 2, \ldots, M\}$ for some polynomially large M. In this case, [69] showed that the two algorithms are identical up to a vanishingly small failure probability.

2.2.2 Martingale Techniques

Our analysis of a single round of Luby's algorithm relies on analyzing certain associated martingales and using martingale concentration inequalities.

Shifted Martingale. Consider a collection $X_1, X_2, \ldots, X_t$ of boolean random variables and let $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] = p_i$. We are interested in obtaining high probability bounds on the sum $S_t = \sum_{i=1}^{t} X_i$. For example, consider X_i to be the indicator random variable for the event that the ith edge is chosen into the matching. Now clearly, the random variables $\{X_i\}$ are not independent, so we cannot use standard Chernoff bounds. If we let $S_i = \sum_{j=1}^{i} X_j$, then one could hope to use martingale inequalities to get a concentration result for S_i. The challenge here is that the sequence $S_1, \ldots, S_t$ is not necessarily a martingale. Nevertheless, when the p_i are bounded, then we can still utilize martingale concentration inequalities to show that S_t does not deviate too much from its mean by considering the following *shifted random variables*:

$$Y_i = S_i - \mathbb{E}[S_i \mid Y_0, \ldots, Y_{i-1}] + Y_{i-1}$$

Since the sequence $Y_1, \ldots, Y_t$ is a martingale and further has bounded variance, we can use bounded variance martingale concentration bounds on Y_t to give good concentration on S_t as well. This shifting idea is inspired by [69], in which it is used in the context of approximate-maximum independent set. In this work, we generalize this shifting idea to broader scenarios, which we discuss in the full version.

Scaled Martingale. In some parts of our analysis, the shifted martingale trick doesn't suffice for our purposes. Consider a sequence of random variables $S_1, \ldots, S_t$ such that $\mathbb{E}[S_i \mid S_1, \ldots, S_{i-1}] = (1-p)S_{i-1}$ for some fixed $0 < p < 1$. In this scenario, we expect the difference $S_i - S_{i-1}$ to decrease as i increases. It is challenging for the shifted martingale trick to exploit such dynamics. The main reason is that in order for the shifted martingale to give a concentration result, we need the expected value of $S_i - S_{i-1}$ to be bounded by some fixed number, which wouldn't exploit the property that the difference is decreasing over time. To get a concentration result in such scenarios, we use another trick which we refer to as the *scaling trick*. We can obtain a concentration bound for S_t by considering the

following scaled random variables:

$$F_i = \frac{S_i}{(1-p)^{i-1}}$$

Observe that F_i is a martingale. This is because $\mathbb{E}[F_i \mid F_1, \cdots, F_{i-1}] = \frac{1}{(1-p)^{i-1}}\mathbb{E}[S_i \mid F_1, \cdots, F_{i-1}] = \frac{1}{(1-p)^{i-1}}\mathbb{E}[S_i \mid S_1, \cdots, S_{i-1}] = \frac{S_{i-1}}{(1-p)^{i-2}} = F_{i-1}$. Therefore, we can get a concentration result for S_t by getting a concentration result for F_t. The main intuition behind the scaling trick is that it exploits the decreasing difference between S_i and S_{i-1} over time. This is exactly the reason for dividing S_i by $(1-p)^i$. For instance, since $F_1 = S_1$, if we get that F_t doesn't deviate too far from F_1, it would imply that $S_i = (1-p)^i F_i \approx (1-p)^i F_1 = (1-p)^i S_1$, which is exactly where we're expecting S_i to be at step i.

2.2.3 Local Regular-Graph Preservation Lemma

We analyze a single round of Luby's algorithm using the above martingale based techniques. First, we show a lemma with the following local guarantee.

▶ **Lemma 5** (Local Regular-Graph Preservation Lemma - Informal). *Let G be a bipartite Δ-regular graph and suppose we run one round of Luby's algorithm on G and let u be an arbitrary node in G. Then, with probability at least $1-\exp(-\operatorname{poly}(\Delta))$, $\Delta/2 \pm o(\Delta)$ neighbors of u get matched.*

Recall that $N(u)$ is the set of neighbors of node u and $N^2(u)$ is the set of nodes at distance exactly 2 from u. Let A_u be the set of edges between $N(u)$ and $N^2(u)$. To prove Lemma 5, we show that roughly $\Delta/2$ edges from A_u are chosen into the matching with probability at least $1-\exp(-\operatorname{poly}(\Delta))$. While the claim holds trivially in expectation, it is challenging to obtain high probability bounds due to the dependencies between u's neighbors.

To simplify exposition, let E_u denote the set of edges that are at most 3 hops away from u, i.e. $E_u = E \cap \big(\{u\} \times N(u)) \cup (N(u) \times N^2(u)) \cup (N^2(u) \times N^3(u))\big)$. Clearly edges not in E_u do not affect any of the edges in A_u and hence can be ignored, so we restrict the analysis to assume that only edges from E_u arrive in the sequential view of Luby's algorithm.

Let $\mathcal{M}_u \subset A_u$ be the set of matching edges chosen from A_u and our goal is to get high probability upper and lower bounds on $|\mathcal{M}_u|$. Let $X_i \in \{0,1\}$ be an indicator random variable for the event that the ith arriving edge belongs to $\mathcal{M}_u$. Let $q_i = \mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}]$ be the probability that the ith edge is from A_u and none of its neighboring edges have already arrived. One could try now to utilize the shifted martingale trick described above to obtain concentration bounds on $|\mathcal{M}_u| = \sum_i X_i$. However, the main challenge here is that q_i itself is a random variable. Our goal is to analyze q_i using martingales analysis. Roughly speaking, we use the scaled martingale trick discussed above to get concentration results for q_i for all i, which enables us to apply the shifted martingale trick to get the desired bounds on $|\mathcal{M}_u|$.

Analyzing q_i. To analyze q_i, we need to understand how many edges *survive* after the first $i-1$ edges have already arrived. Intuitively, an edge is still surviving in iteration i if neither it nor any of its neighbors was not sampled in the first $i-1$ iterations. Let E_i be the set of surviving edges in A_u at the beginning of the ith iteration. Then we have, $q_i = \frac{|E_i|}{|E_u|-(i-1)} = \frac{|E_i|}{\Delta(|N^2(u)|+1)-(i-1)}$ since a sampled surviving edge is always added to the matching.

The final key property towards the proof. To get high probability bounds on $|E_i|$, we first prove that $\mathbb{E}[|E_i| \mid E_{i-1}] \approx (1-2/k)|E_{i-1}|$. This is exactly the setting where the scaled martingale trick can help us to get a concentration result for $|E_i|$, which paves the way for proving Lemma 5.

2.2.4 Regular-Graph Preservation Lemma

We use Lemma 5 to show that applying a single round of Luby's algorithm on a regular graph and removing the matched nodes along with incident edges results in a graph that is still almost regular.

▶ **Lemma 6** (Regular-Graph Preservation Lemma - Informal). *Let G be a bipartite Δ-regular graph and let G' be the graph obtained by running one round of Luby's algorithm on G and deleting matched nodes and their incident edges. Then all but $o(1)$ fraction of nodes in G' have their degree in the range $\Delta/2 \pm o(\Delta)$ with high probability.*

When Δ is large enough ($\Delta \geq \operatorname{poly}\log n$), Lemma 5 followed by a union bound suffices to show that all nodes have their degree in the range $\Delta/2 \pm o(\Delta)$ with high probability. On the other hand, when Δ is small, then by Lemma 5, the expected number of nodes that do *not* have the requisite degree is only $n \cdot \exp(-\operatorname{poly}(\Delta))$. Further, the degree of a node after a round of Luby's algorithm only depends on $O(\operatorname{poly}(\Delta))$ other nodes. So, we can use a Chernoff-Hoeffding with bounded dependence inequality to argue that all but a $\exp(-\operatorname{poly}(\Delta))$-fraction of nodes have the required degree.

2.2.5 Proof Sketch of Theorem 2

We first argue that running one round of Luby's algorithm on an almost regular graph matches a constant fraction of the nodes with high probability. We note that while this claim is easy to see in expectation, obtaining a high probability bound requires the use of our shifted martingale technique, particularly when the graph becomes only almost regular (instead of fully regular, as in the first iteration). Combined with Lemma 6 that states that the resulting graph remains almost regular, we get that after $O(\log 1/\epsilon)$ rounds, at most ϵ-fraction of nodes remain unmatched.

2.2.6 Extension to Node-Averaged Complexity

We now sketch our result for the node-average complexity of MM given the Regular-Graph Preservation Lemma. We can combine Lemma 6 with the observation that in an (almost) regular graph, every node has a constant probability of being matched (and hence is only expected to survive a constant number of rounds under Luby). Since node-averaged complexity cares about the expected time to match a node, our initial rounds of Luby are consistent with our $O(1)$ node-averaged complexity goal. However, we cannot continue this trick until all nodes are matched, since eventually the graph will become too irregular for us to proceed. Instead, we only use Luby until a $O(1/\log\Delta)$ fraction of nodes remain; after so few nodes remain, it is safe for us to finish by repeatedly calling a previous black box [27] that takes $O(\log\Delta/\log\log\Delta)$ rounds to match a constant fraction of nodes but which can handle general (i.e. irregular) graphs. Similar to our previous argument, nodes are only expected to survive a constant number of black box invocations, but since there are so few nodes left our node-averaged complexity is still $O(1)$. Unlike the Luby phase, it is safe to continue this phase until all nodes are matched since it no longer matters how regular the graph is.

2.3 Overview of Lower Bounds

To prove lower bounds against LOCAL algorithms, we use some ideas from a lower bound construction of Ben-Basat, Kawarabayashi, and Schwartzman [32] that gave $\Omega(1/\epsilon)$ lower bounds in the LOCAL model for $(1+\epsilon)$ maximum matching as well as other approximate graph problems. The critical idea in that proof is that when an r-round LOCAL algorithm is deciding what to do with a node v (e.g. who to match it with), it can only use the local structure of the graph around v; in particular, it can only see the r-hop neighborhood around v. This means that we could cut out this r-hop neighborhood from the graph, put it back in differently, and the algorithm would have to make the same decision on v (or for randomized algorithms, the same distribution on decisions). The proof revolves around designing these r-hop neighborhoods as (symmetrical) gadgets, then showing that a constant number of gadgets will (with constant probability) induce an unmatched node between them. The BKS proof uses a simple path as their gadget which involves $O(r)$ nodes. Hence the algorithm can be shown to have an overall error rate of one error per $O(r)$ nodes, so the critical round threshold to allow for $(1+\epsilon)$-multiplicative approximations is $\Omega(1/\epsilon)$ rounds.

Relative to their result, the main upgrade we want to make is that the counterexample graph(s) should be Δ-regular. It is relatively straightforward to take BKS path gadgets and stick them into a large cycle, recovering their $\Omega(1/\epsilon)$ round lower bound for 2-regular bipartite graphs. The main technical hurdle we overcome is generalizing to higher degree. We know from our upper bounds that there must be some degradation as the degree Δ increases, so the main question is, how much efficiency do we need to lose to burn our excess degree? We design gadgets with $O(\Delta r)$ nodes and asymptotically maintain the original error rate of one error per constant number of gadgets (the cycle proof argues about the outcome of two gadgets inducing a mistake, but for the general case we reason about the outcome of five gadgets), yielding $\Omega(1/(\Delta\epsilon))$ round lower bounds. We discuss this formally in the full version.

3 An $O(\log 1/\epsilon)$-round Algorithm

In this section, we show that there is an $O(\log(1/\epsilon))$-round algorithm to find a $(1+\epsilon)$-approximate maximum matching in Δ-regular graphs when $\Delta \geq (\frac{1}{\epsilon})^c$ for a large constant c. Due to space constraints, we defer the pseudocode for the algorithm and full proofs to the full version. We note that the proof makes no attempt to optimize the constant c and we expect that it can be reduced significantly by a more careful analysis.

▶ **Theorem 2** (Main Result I). *Let $c = 10^5$ and let $\epsilon \in (0,1)$ be an accuracy parameter. For Δ-regular graphs with $\Delta > (1/\epsilon)^c$, there is an $O(\log(1/\epsilon))$-round CONGEST algorithm that finds a $(1+\epsilon)$-approximate maximum matching with high probability.*

A Roadmap for the Proof of Theorem 2. The algorithm for proving Theorem 2 first runs a simple color coding step to find a bipartite almost regular subgraph, and then runs Luby's algorithm for $O(\log(1/\epsilon))$ rounds. Since our analysis for Luby's algorithm uses several martingale inequalities, it is cleaner to work with the sequential view of Luby that we present in Section 3.1. In Section 3.2 we show that a single round of Luby's algorithm in an almost regular graph matches a constant fraction of the nodes, with high probability.[7] In Section 3.3, we state our key Regular-Graph Preservation Lemma that shows that running one round

[7] In fact, we prove a more general claim where it suffices that a constant fraction of the edges are balanced.

of Luby's algorithm on an almost regular graph and deleting the matched nodes yields an almost regular graph. A local version of that lemma, which bounds the probability that the degree of a particular node u almost exactly halves after each round is the most technical part of the proof and is given in the full version. Finally, in Section 3.4, we put these components together to finish the proof.

3.1 Sequential View of Luby's Algorithm

In the traditional distributed implementation of one round of Luby's algorithm, each edge e picks a uniformly random integer r_e in the set $\{1, 2, \ldots, M\}$ for an appropriately chosen polynomially large M. An edge e is chosen to be in the matching if $r_e < r_{e'}$ for all neighboring edges e'.

However, it is convenient for our analysis to work with a sequential view of Luby's algorithm. In the sequential view, the edges are sequentially sampled independently without replacement. When an edge e is sampled, it is added to the matching only if none of its neighboring edges had been sampled earlier. Crucially, unlike the greedy matching algorithm, a sampled edge e is *blocked* by a neighboring edge e' that was previously sampled even if e' itself is not in the matching. [69] showed that the two algorithms are equivalent by showing that they produce the same distribution over matchings with high probability[8]. We formalize this in the following proposition.

▶ **Proposition 7** (Proposition 3 in [69]). *For any unweighted graph G with m edges and constant $c' > 0$, let $\mathcal{D}_0$ and $\mathcal{D}_1$ be the distributions over matchings produced by the traditional and sequential views of Luby respectively. The total variation distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ is at most $1/m^{c'}$; in particular*

$$\sum_{matchings\ \mathcal{M}_0} |\mathbb{P}_{\mathcal{M}\sim\mathcal{D}_0}[\mathcal{M} = \mathcal{M}_0] - \mathbb{P}_{\mathcal{M}\sim\mathcal{D}_1}[\mathcal{M} = \mathcal{M}_0]| \leq 1/m^{c'}$$

Following Proposition 7, in this paper we focus on analyzing the sequential view of Luby whenever we want to make claims about a single round of Luby's algorithm. This incurs an additional tiny $1/\operatorname{poly}(n)$ failure probability, which we can tolerate.

3.2 Analyzing One Round of Luby's Algorithm on Almost Regular Graphs

In this section we show that a single round of Luby's algorithm on almost regular graphs matches a constant fraction of the nodes with high probability. In fact, we prove this claim for a more general family of graphs in the following theorem.

▶ **Theorem 8.** *Let G be an undirected graph with n nodes and m edges, and let $d = 2m/n$ be the average degree. Let $E^{low} = \{(u, v) \in E(G) \mid deg(u) \leq 2d, deg(v) \leq 2d\}$ be the set of edges induced by nodes with degree at most $2d$. If $|E^{low}| \geq m/2$, then one round of Luby's algorithm finds a matching in G of size at least $n/288$ with high probability.*

At a high level, the proof of this theorem does the following. Consider the sequential view of Luby's algorithm. When an edge $e = \{u, v\}$ is sampled, it blocks at most $deg(u) + deg(v)$ other edges from joining the matching in the future. However, it can only block at most

[8] They stated the claim in the context of independent sets.

$2(2d) = 4d$ edges from E^{low}, because of the fact that each endpoint of an edge in E^{low} has degree at most $2d$. Thus, in the first Cn sampling steps of sequential Luby for some sufficiently small constant $C < 1$, at most $(4d)(Cn) = 4Cdn = 8Cm$ edges are blocked, meaning that the probability that any particular edge sampled during one of these steps is in E^{low} and has not been blocked is at least $(|E^{low}| - (8Cm))/(m - Cn) > 1/6$ as long as $C < 1/16$. Thus, the found matching has size at least $(Cn)(1/6)$ in expectation. Using martingale concentration inequalities, we can argue that the size of the matching found is at least $(1/2)(Cn)(1/6) > n/288$ with high probability, as desired. We give a formal version of this proof in the full version.

3.3 Regular-Graph Preservation

We first define some notation to facilitate the rest of the discussion. Intuitively, we say a node (α, Δ)-regular if all nodes in its two hop neighborhood have the same degree.

▶ **Definition 9** ((α, Δ)-regular node). *Let Δ be an integer and $\alpha \in [0, 1]$ be a real number. Given a graph G, we say that a node u is (α, Δ)-regular in G if for any $v \in \{u\} \cup N(u) \cup N^2(u)$, we have $\Delta(1 - \alpha) \leq deg(v) \leq \Delta(1 + \alpha)$.*

We can now present our main technical lemma that states that if u is a (α, Δ)-regular node, then its degree becomes almost exactly $\Delta/2$ after running one round of Luby's algorithm, with failure probability exponentially small in Δ. The proof of Lemma 10 is the most technical part of the paper and is deferred to the full version.

▶ **Lemma 10** (Local Regular-Graph Preservation Lemma). *Let $\Delta \geq 2^{10}$ be an integer, and $\alpha \in [0, 1/10]$ be a real number. Let G be a bipartite graph and u be an (α, Δ)-regular node in it. Let $deg'(u)$ be the number of unmatched neighbors of u after running sequential Luby on G. With probability at least $1 - \exp(-\Delta^{1/16})$, it holds that:*

$$\frac{\Delta}{2}\left(1 - (10\alpha + \Delta^{-1/600})\right) \leq deg'(u) \leq \frac{\Delta}{2}\left(1 + (10\alpha + \Delta^{-1/600})\right)$$

As long as Δ is large enough ($\approx \log n$), Lemma 10 followed by a union bound suffices to show that an almost regular graph remains almost regular with their degrees halved after one round of Luby's algorithm. However, for smaller Δ, we can no longer rely on a simple union bound. In the following lemma, we use the Chernoff-Hoeffding concentration inequality with bounded dependence to show that even for small Δ, *almost all* nodes halve their degree.

▶ **Lemma 11** (Regular-Graph Preservation Lemma). *Let $n \geq K \geq \Delta \geq 2^{10}$ be integers, and let $\alpha \in [0, 1/10]$, $\delta \in [0, 1/100]$ be real numbers. Let G be a bipartite graph with n nodes and max degree K and let G' be the graph obtained by running sequential Luby on G and removing all matched nodes together with their incident edges.*

If at least $(1 - \delta)$-fraction of nodes in G are (α, Δ)-regular, then at least $(1 - \delta')$-fraction of nodes in G' are $(\alpha', \Delta/2)$-regular with probability at least $1 - \exp\left(-n/(K^{10}\exp\left(\Delta^{1/99}\right))\right)$, where $\delta' = K^2 \cdot (\delta + 2\exp\left(-\Delta^{1/100}\right)$ and $\alpha' = 10\alpha + \Delta^{-1/600}$.

Proof. We say that a node v is *good* if its degree in G' is in the range $\frac{\Delta}{2}(1 \pm \alpha')$. Any node that's not good is called *bad*. Let R be the set of nodes that are (α, Δ)-regular in G. By Lemma 10, each (α, Δ)-regular node in G that remains unmatched is good with probability at least $1 - \exp\left(-\Delta^{1/16}\right)$. Hence, the expected number of nodes from R that are bad is at most $|R| \cdot \exp\left(-\Delta^{1/16}\right) \leq n$. To obtain a high probability bound, we use Chernoff-Hoeffding inequality with bounded dependence as follows.

For each node $u \in R$, let $B_u \in \{0,1\}$ be an indicator random variable that indicates whether u is bad. We have $\sum_{u\in R} \mathbb{E}[X_u] \leq |R| \cdot \exp\left(-\Delta^{1/16}\right) \leq n \cdot \exp\left(-\Delta^{1/16}\right)$. Since the maximum degree in G is K, each X_u depends on fewer than K^{10} nodes . Hence we apply Chernoff with bounded dependence (see full version for theorem statement) to get:

$$\begin{aligned}
\mathbb{P}\left[\sum_{u\in R} X_u > 2n\exp(-\Delta^{1/100})\right] &\leq \mathbb{P}\left[\sum_{u\in R} X_u > 2|R|\exp(-\Delta^{1/16})\right] \\
&\leq \mathbb{P}\left[\sum_{u\in R} X_u > \sum_{u\in R}\mathbb{E}[X_u] + |R|\exp(-\Delta^{1/16})\right] \\
&\leq \exp\left(-\frac{2(|R|\cdot\exp(-\Delta^{1/16}))^2}{|R|\cdot K^{10}}\right) \\
&= \exp\left(-\frac{2|R|}{K^{10}\exp\left(2\Delta^{1/16}\right)}\right) \\
&\leq \exp\left(-\frac{2|R|}{K^{10}\exp\left(\Delta^{1/99}\right)}\right) \leq \exp\left(-\frac{n}{K^{10}\exp\left(\Delta^{1/99}\right)}\right)
\end{aligned}$$

where the last line used $\Delta > 2^{10}$ and $|R| \geq (1-\delta)n \geq n/2$. We say that a node v poisons a node u if v prevents u from being $(\alpha', \Delta/2)$-regular, i.e., v poisons u if v is *bad* and is at distance at most 2 from u. We note that each bad node poisons at most K^2 nodes. Let B be the set of bad nodes, i.e., we have $|B| = \sum_{u\in R} X_u$ and let $\tilde{B} = V(G) \setminus R$ be the set of nodes that are not (α, Δ) regular in G. For nodes in $\tilde{B}$, we have no guarantee on the probability of whether they become good, so we always assume that they poison K^2 nodes. In total, the number of nodes that are not $(\alpha', \Delta/2)$-regular in G' is at most $(|\tilde{B}| + |B|)K^2$ which is at most:

$$\begin{aligned}
(|\tilde{B}| + 2n\cdot\exp\left(-\Delta^{1/100}\right))\cdot K^2 &\leq (\delta n + 2n\cdot\exp\left(-\Delta^{1/100}\right))\cdot K^2 \\
&= n\cdot K^2\cdot(\delta + 2\exp\left(-\Delta^{1/100}\right))
\end{aligned}$$

with probability at least $1 - \exp\left(-n/(K^{10}\exp\left(\Delta^{1/99}\right))\right)$, as desired. ◀

Lemma 11 shows that after each round of Luby's algorithm, the remaining graph still satisfies the requirement that most vertices remain almost regular, albeit with worsening parameters. The following technical claim shows that these parameters remain small enough after $O(\log(1/\epsilon)$ rounds. The proof uses basic arithmetic and is deferred to the appendix.

▷ Claim 12. Let Δ be an integer, $c = 1/10^5$, and ϵ be a real number satisfying $1 > \epsilon \geq 1/\Delta^c$. Furthermore let:

1. $\alpha_0 = \Delta^{-1/600}$ and $\alpha_i = 10\alpha_{i-1} + \Delta^{-1/600}$ for $i \geq 1$,
2. $\delta_0 = \exp(-\Delta^{1/200})$, and $\delta_i = \Delta^2(\delta_{i-1} + 2\exp(-(\Delta/2^i)^{1/100}))$ for $i \geq 1$.

It holds that $\alpha_i \leq 1/10$ and $\delta_i \leq \exp(-\Delta^{1/300})$ for $1 \leq i \leq 10\log(1/\epsilon)$.

We now extend Lemma 11 to a multi-round argument, showing that most of nodes' degrees after running Luby's algorithm for i rounds become $\approx \Delta/2^i$.

▶ **Lemma 13** (Multi-Round Regular-Graph Preservation). *Let $(\Delta, \epsilon, (\alpha_i), (\delta_i))$ be as defined in Claim 12. Let G be a bipartite n-node graph, where all but $\delta = \delta_0 = \exp\left(-\Delta^{1/200}\right)$-fraction of the nodes are (α_0, Δ)-regular. For $i \leq 10\log(1/\epsilon)$, let G_i be the graph obtained after running Luby's algorithm for i rounds on G, where in each round we remove the matched nodes together with their incident edges. If $|V(G_i)| \geq \epsilon n$, then with high probability, at least $(1-\delta_i)$-fraction of the nodes in G_i are $(\alpha_i, \Delta/2^i)$-regular.*

Proof. The idea is to use a recursive argument where we apply Lemma 10 with a simple union bound in the dense case, and Lemma 11 in the sparse case. For convenience of notation, let $\Delta_i = \Delta/2^i$. Since we apply Lemmas 10 and 11 recursively for i rounds, we need to ensure that $\alpha_i \leq 1/10$ and $\delta_i \leq 1/100$, which follows from Claim 12. The proof is split into two cases, a dense case where $\Delta \geq \log^{50} n$, and a sparse case. We start with the dense case.

Dense case where $\Delta \geq \log^{50} n$. By Lemma 10 and a simple union bound argument, in the case where $\Delta \geq \log^{50} n$, *all* the nodes are $(\alpha_i, \Delta/2^i)$-regular after running Luby's algorithm for i rounds with high probability. This is because the failure probability of Lemma 10 is at most $\exp(-\Delta_i^{1/16})$ at round i, and $\Delta_i \geq \Delta^{0.99}$ for any i (since $\epsilon \geq 1/\Delta^{1/c}$ and $i \leq 10\log(1/\epsilon)$).

Sparse case where $\Delta \leq \log^{50} n$. The idea is to apply Lemma 11 recursively for i rounds and the proof follows by induction.

Induction base case i=1. By Proposition 7, traditional and sequential Luby produce the same distributions over matchings in the graph G, up to $1/\operatorname{poly}(n)$ total variation distance. Hence, for $i = 1$ the claim follows directly from Lemma 11, since the failure probability of Lemma 11 is only $\exp\left(-n/\Delta^{10}\exp\left(\Delta^{1/99}\right)\right) < 1/n^{1000}$, for $\Delta \leq \log^{50} n$.

Induction step. Assume that the claim is true for i, i.e., assume that all but $|V(G_i)|\cdot\delta_i$ nodes in G_i are $(\alpha_i, \Delta/2^i)$-regular with high probability. We show that the claim is true for $i+1$. Again, by Proposition 7, traditional Luby and sequential Luby produce the same distributions over matchings in the graph G_i, up to a $1/|E(G_i)|^c \leq 1/(|V(G_i)|)^c \leq 1/(\epsilon n)^c \leq (1/n^{0.99})^c$ total variation distance, for an arbitrarily large constant c. Since the max degree in G_i is at most Δ, by Lemma 11 it holds that all but $|V(G_{i+1})|\cdot\delta_{i+1}$ nodes in G_{i+1} are $(\alpha_{i+1}, \Delta/2^{i+1})$-regular in G_{i+1} with probability at least

$$1 - \exp\left(-\frac{|V(G_i)|}{\Delta^{10}\exp\left(\Delta_i^{1/99}\right)}\right) \geq 1 - \exp\left(-\frac{\epsilon n}{\Delta^{10}\exp\left(\Delta^{1/99}\right)}\right) \geq 1 - \frac{1}{n^{1000}}$$

where the first inequality holds since $\Delta_i \leq \Delta$ and $|V(G_i)| \geq \epsilon n$, and the second inequality holds since $\epsilon \geq 1/\Delta^{1/100} \geq 1/n^{1/100}$ and $\Delta \leq \log^{50} n$. Hence, by a union bound on all i's, we get that as long as $V(G_i)$ has at least ϵn nodes, it holds that all but $|V(G_i)|\cdot\delta_i$ nodes are (α_i, Δ_i)-regular in G_i with probability at least $1 - 1/n^{100}$. ◀

3.4 Putting it together

Lemma 13 shows that the graph remains almost regular after $i \approx \log(1/\epsilon)$ repeated applications of Luby's algorithm, and Theorem 8 showed that each round of Luby's on an almost regular graph matches a constant fraction of nodes. We can now combine these two results to show that as long as the remaining graph is large enough, each application of Luby's algorithm matches a constant fraction of nodes.

We first present a corollary of Theorem 8 to formalize that the almost regular graphs implied by Lemma 13 do indeed satisfy the requirements of Theorem 8. We defer the proof to the appendix for brevity.

▶ **Corollary 14.** *Let $\Delta \geq C$, where C is a large enough constant, and let G' be a graph with n' nodes and max degree Δ. Suppose at least $(1-\exp(-\Delta^{1/300}))$-fraction of the nodes are (α', Δ')-regular for $\alpha' \leq 1/10$. It holds that Luby's algorithm matches $n'/288$ nodes in G' with high probability.*

We are now ready to prove Theorem 2.

Proof of Theorem 2. We first provide a proof that assumes that the graph in bipartite, and then we lift this assumption by a simple sampling argument. We simply run Luby's algorithm in G for $i = 10\log(1/\epsilon)$ rounds. By Lemma 13, all but an $\exp(-\Delta^{1/300})$-fraction of the nodes in the graph are (α_i, Δ_i)-regular at round i with high probability. Hence, by Corollary 14, in each of these rounds we match at least a $(1/288)$-fraction of the nodes. Therefore, after $O(\log(1/\epsilon))$, all but ϵ-fraction of the nodes are matched, as desired.

Overcoming bipartiteness. Lemma 13 assumes that the input graph is bipartite. To overcome this, we apply a simple color-coding trick. Before running Luby's algorithm, each node picks a uniformly random color independently in $\{0, 1\}$. This naturally defines a bipartite subgraph graph G' by ignoring all monocolored edges. Observe that for a given node $u \in G$, the degree of u in G' is $(\Delta/2)(1 \pm \Delta^{-0.4})$ with probability at least $1 - \exp(-\Delta^{0.2}/6)$ by a standard Chernoff-Hoeffding bound. Hence, if $\Delta \geq \log^{50} n$, then all the nodes degrees in G' are $\Delta(1 \pm \Delta^{-0.4})$ with high probability. Therefore, all the nodes are also $(\Delta^{-0.4}, \Delta)$-regular in that case. Otherwise, if $\Delta < \log^{50} n$, then we can use a bounded dependence Chernoff-Hoeffding type argument, as follows. By a union bound, the probability that a node u isn't $(\Delta^{-0.4}, \Delta)$-regular is at most $\Delta^2 \cdot \exp(-\Delta^{0.2}/6) \leq \exp(-\Delta^{0.1})$. Hence, the expected number of nodes that aren't $(\Delta^{-0.4}, \Delta)$-regular is at most $n \cdot \exp(-\Delta^{0.1})$. Furthermore, the event of whether a node is $(\Delta^{-0.4}, \Delta)$-regular depends only on $< \Delta^{10}$ other nodes. Hence, by applying Chernoff with bounded dependence (see full version for details), we get that with high probability, all but $n \cdot \exp(-\Delta^{1/200})$ nodes are $(\Delta^{-0.4}, \Delta)$-regular in G'. Hence, since $\Delta^{-0.4} \leq \Delta^{-1/600}$, it suffices to apply Lemma 13 on G'. This concludes the proof. ◀

References

1 Mohamad Ahmadi and Fabian Kuhn. Distributed maximum matching verification in CONGEST. In *DISC*, volume 179 of *LIPIcs*, pages 37:1–37:18, 2020. doi:10.4230/LIPICS.DISC.2020.37.

2 Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. Distributed approximate maximum matching in the CONGEST model. In *DISC*, volume 121 of *LIPIcs*, pages 6:1–6:17, 2018. doi:10.4230/LIPICS.DISC.2018.6.

3 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

4 Noga Alon, Sonny Ben-Shimon, and Michael Krivelevich. A note on regular ramsey graphs. *J. Graph Theory*, 64(3):244–249, 2010. doi:10.1002/JGT.20453.

5 Noga Alon, Amin Coja-Oghlan, Hiêp Hàn, Mihyun Kang, Vojtech Rödl, and Mathias Schacht. Quasi-randomness and algorithmic regularity for graphs with general degree distributions. *SIAM J. Comput.*, 39(6):2336–2362, 2010. doi:10.1137/070709529.

6 Noga Alon, Shmuel Friedland, and Gil Kalai. Every 4-regular graph plus an edge contains a 3-regular subgraph. *J. Comb. Theory B*, 37(1):92–93, 1984. doi:10.1016/0095-8956(84)90048-0.

7 Noga Alon, Shmuel Friedland, and Gil Kalai. Regular subgraphs of almost regular graphs. *J. Comb. Theory B*, 37(1):79–91, 1984. doi:10.1016/0095-8956(84)90047-9.

8 Noga Alon and Guy Moshkovitz. Limitations on regularity lemmas for clustering graphs. *Adv. Appl. Math.*, 124:102135, 2021. doi:10.1016/J.AAM.2020.102135.

9 Noga Alon and Pawel Pralat. Modular orientations of random and quasi-random regular graphs. *Comb. Probab. Comput.*, 20(3):321–329, 2011. doi:10.1017/S0963548310000544.

10 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, volume 107 of *LIPIcs*, pages 7:1–7:16, 2018. doi:10.4230/LIPICS.ICALP.2018.7.

11 Armen S. Asratian and Nikolai N. Kuzjurin. On the number of nearly perfect matchings in almost regular uniform hypergraphs. *Discret. Math.*, 207(1-3):1–8, 1999.

12 Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. In *STOC*, pages 131–144, 2023. doi:10.1145/3564246.3585110.

13 Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *FOCS*, pages 909–932, 2023. doi:10.1109/FOCS57990.2023.00058.

14 László Babai. On the complexity of canonical labeling of strongly regular graphs. *SIAM J. Comput.*, 9(1):212–216, 1980. doi:10.1137/0209018.

15 László Babai. On the automorphism groups of strongly regular graphs I. In *ITCS*, pages 359–368. ACM, 2014. doi:10.1145/2554797.2554830.

16 László Babai, Xi Chen, Xiaorui Sun, Shang-Hua Teng, and John Wilmes. Faster canonical forms for strongly regular graphs. In *FOCS*, pages 157–166. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.25.

17 Alkida Balliu, Thomas Boudier, Sebastian Brandt, and Dennis Olivetti. Tight lower bounds in the supported LOCAL model. In *PODC*, pages 95–105. ACM, 2024. doi:10.1145/3662158.3662798.

18 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. The distributed complexity of locally checkable problems on paths is decidable. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 262–271. ACM, 2019. doi:10.1145/3293611.3331606.

19 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, and Jukka Suomela. Efficient classification of locally checkable problems in regular trees. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.DISC.2022.8.

20 Alkida Balliu, Sebastian Brandt, Yi-Jun Chang, Dennis Olivetti, Jan Studený, Jukka Suomela, and Aleksandr Tereshchenko. Locally checkable problems in rooted trees. *Distributed Comput.*, 36(3):277–311, 2023. doi:10.1007/S00446-022-00435-9.

21 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. *J. ACM*, 68(5):39:1–39:30, 2021. doi:10.1145/3461458.

22 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Improved distributed lower bounds for MIS and bounded (out-)degree dominating sets in trees. In *PODC*, pages 283–293. ACM, 2021. doi:10.1145/3465084.3467901.

23 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed Δ-coloring plays hide-and-seek. In *STOC*, pages 464–477. ACM, 2022. doi:10.1145/3519935.3520027.

24 Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed maximal matching and maximal independent set on hypergraphs. In *SODA*, pages 2632–2676. SIAM, 2023. doi:10.1137/1.9781611977554.CH100.

25 Alkida Balliu, Sebastian Brandt, and Dennis Olivetti. Distributed lower bounds for ruling sets. *SIAM J. Comput.*, 51(1):70–115, 2022. doi:10.1137/20M1381770.

26 Alkida Balliu, Mohsen Ghaffari, Fabian Kuhn, and Dennis Olivetti. Node and edge averaged complexities of local graph problems. *Distributed Comput.*, 36(4):451–473, 2023. doi:10.1007/S00446-023-00453-1.

27 Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *PODC*, pages 165–174, 2017. doi:10.1145/3087801.3087806.

28 Reuven Bar-Yehuda, Keren Censor-Hillel, and Gregory Schwartzman. A distributed $(2+\epsilon)$-approximation for vertex cover in $O(\log\delta/\epsilon \log\log \delta)$ rounds. In *PODC*, pages 3–8. ACM, 2016. doi:10.1145/2933057.2933086.

29 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.60.

30 Leonid Barenboim and Yaniv Tzur. Distributed symmetry-breaking with improved vertex-averaged complexity. *CoRR*, abs/1805.09861, 2018. arXiv:1805.09861.

31 Ran Ben-Basat, Guy Even, Ken-ichi Kawarabayashi, and Gregory Schwartzman. Optimal distributed covering algorithms. *Distributed Comput.*, 36(1):45–55, 2023. doi:10.1007/S00446-021-00391-W.

32 Ran Ben-Basat, Ken-ichi Kawarabayashi, and Gregory Schwartzman. Parameterized distributed algorithms. In *DISC*, 2019.

33 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. In *FOCS*, pages 1563–1588, 2023.

34 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *SODA*, pages 100–128, 2023. doi:10.1137/1.9781611977554.CH5.

35 Sebastian Brandt. An automatic speedup theorem for distributed problems. In *PODC*, pages 379–388. ACM, 2019. doi:10.1145/3293611.3331611.

36 Sebastian Brandt, Orr Fischer, Juho Hirvonen, Barbara Keller, Tuomo Lempiäinen, Joel Rybicki, Jukka Suomela, and Jara Uitto. A lower bound for the distributed lovász local lemma. In *STOC*, pages 479–488. ACM, 2016. doi:10.1145/2897518.2897570.

37 Sebastian Brandt, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Patric R. J. Östergård, Christopher Purcell, Joel Rybicki, Jukka Suomela, and Przemyslaw Uznanski. LCL problems on grids. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 101–110. ACM, 2017. doi:10.1145/3087801.3087833.

38 Sebastian Brandt, Yannic Maus, Ananth Narayanan, Florian Schager, and Jara Uitto. On the locality of hall's theorem. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 4198–4226. SIAM, 2025. doi:10.1137/1.9781611978322.143.

39 Sebastian Brandt and Dennis Olivetti. Truly tight-in-Δ bounds for bipartite maximal matching and variants. In Yuval Emek and Christian Cachin, editors, *PODC*, pages 69–78. ACM, 2020. doi:10.1145/3382734.3405745.

40 Niv Buchbinder, Joseph (Seffi) Naor, and David Wajc. Lossless online rounding for online bipartite matching (despite its impossibility). In *SODA*, pages 2030–2068, 2023. doi:10.1137/1.9781611977554.CH78.

41 Marc Bury, Elena Grigorescu, Andrew McGregor, Morteza Monemizadeh, Chris Schwiegelshohn, Sofya Vorotnikova, and Samson Zhou. Structural results on matching estimation with applications to streaming. *Algorithmica*, 81(1):367–392, 2019. doi:10.1007/S00453-018-0449-Y.

42 Teena Carroll, David J. Galvin, and Prasad Tetali. Matchings and independent sets of a fixed size in regular graphs. *J. Comb. Theory A*, 116(7):1219–1227, 2009. doi:10.1016/J.JCTA.2008.12.008.

43 Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. *SIAM J. Comput.*, 48(1):122–143, 2019. doi:10.1137/17M1117537.

44 Soumyottam Chatterjee, Robert Gmyr, and Gopal Pandurangan. Sleeping is efficient: MIS in $O(1)$-rounds node-averaged awake complexity. In Yuval Emek and Christian Cachin, editors, *PODC*, pages 99–108. ACM, 2020. doi:10.1145/3382734.3405718.

45 Andrzej Czygrinow and Michal Hanckowiak. Distributed algorithm for better approximation of the maximum matching. In *COCOON*, volume 2697 of *Lecture Notes in Computer Science*, pages 242–251, 2003. doi:10.1007/3-540-45071-8_26.

46 Guy Even, Moti Medina, and Dana Ron. Distributed maximum matching in bounded degree graphs. In *ICDCN*, pages 18:1–18:10, 2015. doi:10.1145/2684464.2684469.

47 Laurent Feuilloley. How long it takes for an ordinary node with an ordinary ID to output? In Shantanu Das and Sébastien Tixeuil, editors, *SIROCCO*, volume 10641 of *Lecture Notes in Computer Science*, pages 263–282. Springer, 2017. doi:10.1007/978-3-319-72050-0_16.

48 Manuela Fischer. Improved deterministic distributed matching via rounding. *Distributed Comput.*, 33(3-4):279–291, 2020. doi:10.1007/S00446-018-0344-4.

49 Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 180–191. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.25.

50 Manuela Fischer, Slobodan Mitrovic, and Jara Uitto. Deterministic $(1+\epsilon)$-approximate maximum matching with poly$(1/\epsilon)$ passes in the semi-streaming model and beyond. In *STOC*, pages 248–260, 2022. doi:10.1145/3519935.3520039.

51 Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *SODA*, pages 270–277. SIAM, 2016. doi:10.1137/1.9781611974331.ch20.

52 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved massively parallel computation algorithms for MIS, matching, and vertex cover. In *PODC*, pages 129–138. ACM, 2018. doi:10.1145/3212734.3212743.

53 Mohsen Ghaffari and Christoph Grunau. Faster deterministic distributed MIS and approximate matching. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1777–1790. ACM, 2023. doi:10.1145/3564246.3585243.

54 Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2148–2179. IEEE, 2024. doi:10.1109/FOCS61266.2024.00007.

55 Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *FOCS*, pages 662–673, 2018. doi:10.1109/FOCS.2018.00069.

56 Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In *STOC*, pages 418–430, 2018. doi:10.1145/3188745.3188906.

57 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in O(n log n) time in regular bipartite graphs. *SIAM Journal on Computing*, 42(3):1392–1404, 2013. doi:10.1137/100812513.

58 Louis Golowich. A new berry-esseen theorem for expander walks. In *STOC*, pages 10–22. ACM, 2023. doi:10.1145/3564246.3585141.

59 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In *SODA*, pages 1886–1898, 2019. doi:10.1137/1.9781611975482.114.

60 Christoph Grunau, Václav Rozhon, and Sebastian Brandt. The landscape of distributed complexities on trees and beyond. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 37–47. ACM, 2022. doi:10.1145/3519270.3538452.

61 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *ICALP*, volume 132 of *LIPIcs*, pages 67:1–67:14, 2019. doi:10.4230/LIPICS.ICALP.2019.67.

62 Manoj Gupta and Richard Peng. Fully dynamic (1+ ϵ)-approximate matchings. In *FOCS*, pages 548–557, 2013. doi:10.1109/FOCS.2013.65.

63 David G. Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. In *FOCS*, pages 700–724, 2019. doi:10.1109/FOCS.2019.00048.

64 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

65 Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Inf. Process. Lett.*, 22(2):77–80, 1986. doi:10.1016/0020-0190(86)90144-4.

66 Taisuke Izumi, Naoki Kitamura, and Yutaro Yamaguchi. A nearly linear-time distributed algorithm for exact maximum matching. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4062–4082. SIAM, 2024. doi:10.1137/1.9781611977912.141.

67 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *STOC*, pages 587–596, 2011. doi:10.1145/1993636.1993715.

68 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990. doi:10.1145/100216.100262.

69 Ken-ichi Kawarabayashi, Seri Khoury, Aaron Schild, and Gregory Schwartzman. Improved distributed approximations for maximum independent set. In *DISC*, volume 179 of *LIPIcs*, pages 35:1–35:16, 2020. doi:10.4230/LIPICS.DISC.2020.35.

70 Ludek Kucera. Canonical labeling of regular graphs in linear average time. In *FOCS*, pages 271–279. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.11.

71 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA*, pages 980–989. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109666.

72 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, 2016. doi:10.1145/2742012.

73 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

74 Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):38:1–38:17, 2015. doi:10.1145/2786753.

75 Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed approximate matching. *SIAM J. Comput.*, 39(2):445–460, 2009. doi:10.1137/080714403.

76 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

77 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *FOCS*, pages 253–262, 2013. doi:10.1109/FOCS.2013.35.

78 Andrew McGregor. Finding graph matchings in data streams. In *APPROX*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181, 2005. doi:10.1007/11538462_15.

79 Aranyak Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2013. doi:10.1561/0400000057.

80 Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM J. Discret. Math.*, 4(3):409–412, 1991. doi:10.1137/0404036.

81 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

82 Ami Paz and Gregory Schwartzman. A (2+ϵ)-approximation for maximum weight matching in the semi-streaming model. *ACM Trans. Algorithms*, 15(2):18:1–18:15, 2019. doi:10.1145/3274668.

83 David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, USA, 2000.

84 Václav Rozhon. Invitation to local algorithms. *CoRR*, abs/2406.19430, 2024. doi:10.48550/arXiv.2406.19430.

85 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

86 Daniel A. Spielman. Faster isomorphism testing of strongly regular graphs. In *STOC*, pages 576–584. ACM, 1996. doi:10.1145/237814.238006.

87 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. doi:10.1145/2431211.2431223.

88 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *FOCS*, pages 919–930, 2020. doi:10.1109/FOCS46700.2020.00090.

89 Raphael Yuster. Maximum matching in regular and almost regular graphs. *Algorithmica*, 66(1):87–92, 2013. doi:10.1007/S00453-012-9625-7.

90 Or Zamir. Algorithmic applications of hypergraph and partition containers. In *STOC*, pages 985–998. ACM, 2023. doi:10.1145/3564246.3585163.

Natural Calamities Demand More Rescuers: Exploring Connectivity Time Dynamic Graphs

Ashish Saxena
Indian Institute of Technology Ropar, Rupnagar, Punjab, India

Kaushik Mondal
Indian Institute of Technology Ropar, Rupnagar, Punjab, India

Abstract

We study the exploration problem by mobile agents in Connectivity Time dynamic graphs. The Connectivity Time model was introduced by Michail et al. [JPDC 2014] and is arguably one of the weakest dynamic graph connectivity models. We prove that exploration is impossible in such graphs using $\frac{(n-1)(n-2)}{2}$ mobile agents starting from an arbitrary initial configuration, even when agents have full knowledge of system parameters, global communication, full visibility, and infinite memory. We then present an exploration algorithm that uses $\frac{(n-1)(n-2)}{2} + 1$ agents equipped with global communication, 1-hop visibility and $O(\log n)$ memory.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Mobile agents, Anonymous graphs, Exploration, Dynamic graphs, Deterministic algorithm

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.41

Funding *Ashish Saxena*: Acknowledge the financial support from IIT Ropar.
Kaushik Mondal: Acknowledge the ISIRD grant provided by IIT Ropar.

1 Introduction

The exploration of graphs by mobile agents is a well-studied problem in distributed computing and has foundational importance in theoretical computer science. Originating from early work by Shannon [39], the objective is for mobile agents to collectively visit every node in a given network. Depending on the requirements, the task may involve visiting each node at least once (exploration with termination) or repeatedly over time (perpetual exploration). This problem is not only of theoretical interest but also has practical implications for systems involving autonomous agents, such as robots, software agents, or vehicles, where exploration helps in fault detection, information dissemination, or data collection across the network.

The graph exploration problem has been studied under a wide range of assumptions. These include whether the nodes are uniquely labelled or anonymous, whether agents have distinct identities or are indistinguishable, and the mode of communication or interaction among agents, such as using whiteboards, tokens, face-to-face meetings, or vision-based mechanisms. Variations also arise based on the degree of synchrony among agents (asynchronous, semi-synchronous, or fully-synchronous), the extent of their knowledge about the network, and the amount of memory available to them (refer to [2,6,7,11,13,14,21,22,37], for a comprehensive overview, refer to [9]). Despite the diversity in models, most of the prior research is on static graphs, meaning the graph structure remains fixed throughout the exploration. While this assumption works well for traditional networks, where changes typically result from failures but it falls short in capturing the behaviour of today's highly dynamic networks.

The dynamic nature of modern networks presents significant challenges in addressing various algorithmic problems in mobile computing and related domains, as the underlying network topology evolves over time. From the perspective of mobile agents, this means


© Ashish Saxena and Kaushik Mondal;
licensed under Creative Commons License CC-BY 4.0



that agents must carry out their tasks in an environment that evolves over time steps. A foundational model capturing such dynamic behaviour was introduced by Kuhn et al. [32]. In their framework, they defined a stability property known as T-Interval Connectivity (for $T \geq 1$), which requires that in every sequence of T consecutive rounds, there exists a stable, connected spanning subgraph, although additional edges may appear or disappear in each round. Later, Michail et al. [33] proposed a more relaxed and natural notion of connectivity in dynamic networks, particularly suitable for networks that may be disconnected at individual time steps. They introduced the concept of Connectivity Time, defined as follows:

Let V be a fixed set of nodes and $S = \{(u, v), |, u, v \in V\}$ be the set of possible edges. Let $\mathcal{P}(S)$ denote the power set of S. A synchronous dynamic network is modeled as a dynamic graph $G = (V, E)$, where $E : \mathbb{N} \to \mathcal{P}(S)$ maps each round number $r \in \mathbb{N} \cup \{0\}$ to the set of edges present at that round. The static graph at round r is denoted by $\mathcal{G}_r = (V, E(r))$.

Definition. [33] *The Connectivity Time of a dynamic graph $G = (V, E)$ is the minimum integer $T \in \mathbb{N}$ such that $\forall\, r \in \mathbb{N} \cup \{0\}$, the union graph $G_{r,T} := (V, \bigcup_{i=r}^{r+T-1} E(i))$ is connected.*

This model generalizes T-Interval Connectivity, but unlike T-Interval Connectivity, it allows temporary disconnections. Thus, Connectivity Time is strictly weaker than T-Interval Connectivity. In the next section, we discuss the model and problem definition.

2 Model and Problem Definition

Dynamic graph model. We consider a dynamic network modeled as a sequence of undirected graphs $G = (V, E)$, where the node set V remains fixed over time and satisfies $|V| = n$. Define $S = \{(u, v) \,|\, u, v \in V\}$ as the set of all possible edges, and let $\mathcal{P}(S)$ denote its power set. The function $E : \mathbb{N} \to \mathcal{P}(S)$ maps each round number $r \in \mathbb{N} \cup \{0\}$ to the set of edges $E(r)$ present at that round, yielding the snapshot graph $\mathcal{G}_r = (V, E(r))$. The dynamic graph G is thus given as a sequence $\langle \mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \ldots \rangle$. We assume the presence of a dynamic adversary that may insert or delete any edge at the beginning of each round. The degree of a node v at round r is denoted by $\deg_r(v)$. The diameter of $\mathcal{G}_r$ is denoted by D_r.

Each snapshot graph $\mathcal{G}_r$ is unweighted, undirected, and anonymous. Moreover, the graph is port-labelled: for any node $v \in \mathcal{G}_r$, the incident edges are assigned distinct local port numbers in the range $[0, \deg_r(v) - 1]$. For an edge (u, v), the port numbers at u and v are independently assigned and unrelated. Port labellings can differ across rounds; i.e., port numbers at a node in $\mathcal{G}_r$ may not match those in $\mathcal{G}_{r'}$ for $r \neq r'$. Nodes do not have any storage capability. A node is referred to as **hole** in round r if no agent is present at that node, and as **multinode** if two or more agents occupy it at round r. In this work, the graph $G(= \langle \mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2, \ldots \rangle)$ maintains the Connectivity Time property for some T, i.e., for every $r \geq 0$, the graph $G_{r,T} := (V, \bigcup_{i=r}^{r+T-1} E(i))$ is connected.

Agent model. We consider ℓ mobile agents that are initially placed arbitrarily on the nodes of G. Each agent has a unique identifier from the range $[1, n^c]$, where c is some constant, and knows only its own ID. Agents are equipped with $O(\log n)$ memory and execute the algorithm under a fully synchronous scheduler, i.e., in each round t, every agent executes a Communicate-Compute-Move (CCM) cycle:

- **Communicate:** Agents communicate as per the communication model.
- **Compute:** Based on its local view and any received information, the agent performs computation, including deciding whether and where to move.
- **Move:** The agent moves through a chosen port or stays idle.

The time complexity is measured by the number of synchronous rounds. We refer to $\mathcal{G}_r$ together with the agents' positions as the *configuration*. With a slight abuse of notation, we may denote the configuration at round r by $\mathcal{G}_r$.

Visibility model. We adopt a standard visibility framework where agents have l-hop visibility. In the l-hop model [1,34], at the beginning of round r, an agent can see the subgraph induced by nodes within distance l from its current location in $\mathcal{G}_r$, including the presence or absence of agents in that neighbourhood. When $l = D_r$, this provides full visibility at round r.

Communication model. In this work, we consider the global communication model [8, 20, 30, 31, 36]. The global communication allows agents to exchange messages with any agent located in the same connected component of $\mathcal{G}_r$, utilizing the graph's links. The global communication between two different connected components of $\mathcal{G}_r$ is not possible, as there is no edge between two different connected components.

Problem definition. A node v is visited by round r if at least one agent is at node v at round t, where $t \in [0, r]$. An algorithm achieves *exploration* if every node is visited at least once. And, an algorithm achieves *perpetual exploration* if every node is visited infinitely often.

3 Related Work

Exploration of dynamic graphs has been widely studied in centralized settings, where agents have full knowledge of the network's evolution. Notably, optimal exploration schedules have been analyzed under 1-Interval Connectivity [18] and extended in subsequent works [15–17]. Specific topologies such as rings and cactuses have also been explored under T-Interval and 1-Interval Connectivity, respectively [26, 27].

Distributed exploration, with limited agent knowledge, has received less attention. Probabilistic methods like random walks were introduced in early foundation work [3], while deterministic approaches focus on periodic graphs and carrier models [18,19,27,28]. Perpetual exploration and exploration with termination have been studied in 1-Interval Connected rings using 2 or 3 agents under Fsync and Ssync models [4, 5, 12]. Other results include exploration with $O(n)$ agents in toroidal networks [24], and single-agent strategies with partial foresight [25]. A significant advancement in this area is the work by Gotoh et al. [23], which investigates the fundamental limits of exploration in time-varying graphs under Fsync and Ssync schedulers. In their model, the network is derived from a fixed footprint graph from which edges are deleted dynamically. Agents are able to detect missing edges indirectly when their attempted movements fail. Additionally, port numbers at nodes remain fixed throughout the computation, inherited from the footprint. In contrast, our model assumes the absence of a global footprint. More importantly, port numbers are not fixed over time, as they depend on the local degree of $\mathcal{G}_r$. Consequently, agent movements are always successful, and agents may be unable to detect the change of topology. These characteristics make our model weaker than the one studied in [23], as it places fewer constraints on the dynamic behaviour of the network and provides the agents with less information. In this work, we study perpetual exploration in the Connectivity Time dynamic graphs.

Recently, Saxena et al. [38] studied exploration under various connectivity models, including Connectivity Time, and showed that exploration is impossible with at most n agents (under model assumptions consistent with ours). However, their result did not

yield tight bounds. In this work, we strengthen their impossibility result by proving that exploration is not solvable even with up to $(n-2)(n-1)/2$ agents, and complement this with a matching algorithmic upper bound. Our contributions are presented in the next section.

4 Our Contribution

In this work, we present the following two results:

1. Exploration is impossible in the Connectivity Time dynamic graphs by $\frac{(n-2)(n-1)}{2}$ agents starting from an arbitrary initial configuration, even if agents have infinite memory, full visibility, global communication, and knowledge of all parameters (refer Theorem 4).[1]
2. We present a perpetual exploration algorithm for the Connectivity Time dynamic graphs using $\frac{(n-2)(n-1)}{2}+1$ agents starting from arbitrary initial configuration, where each agent has 1-hop visibility, global communication, and $O(\log n)$ memory (refer Theorem 17).

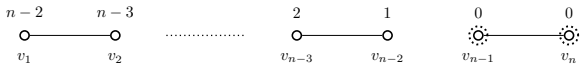


Figure 1 Initial configuration $\mathcal{C}_0$ of $\frac{(n-2)(n-1)}{2}$ agents.

5 Impossibility Result

In this section, we show that exploration is impossible to solve using $\frac{(n-2)(n-1)}{2}$ agents. Before proceeding with the construction of $G = \langle \mathcal{G}_0, \mathcal{G}_1, \ldots \rangle$, we first outline the high-level idea behind the impossibility result.

High-level idea. While there remains a non-empty set of unexplored nodes, the goal is to transfer agents from explored and occupied nodes to those unexplored ones. As it can be difficult to differentiate between an unexplored node with an explored but unoccupied node, the algorithm may require to transfer agents from explored and occupied nodes to explored but currently unoccupied nodes as well. If an algorithm succeeds in keeping all explored nodes occupied, then eventually, as the adversary must pick an edge across the cut, some agent will move to an unexplored node and hence the node becomes visited. However, the adversary is powerful: if b agents move from a node with x agents to a node with $y < x$ agents according to some deterministic algorithm, making the new counts $y' = y + b$ and $x' = x - b$, the adversary can flip the roles of these two nodes in the next graph instance, effectively undoing progress as the algorithm performs the reverse operation. The proofs formalize this idea: with fewer than $(n-1)(n-2)/2 + 1$ agents, the adversary can always choose such a flip whereas with that many agents it cannot always do that.

Dynamic graph G. Let $\boldsymbol{n = 2k}$ for some $k \in \mathbb{N}$, $n \geq 4$ and $T \geq 2$. We give an initial configuration with $\frac{(n-2)(n-1)}{2}$ agents such that the exploration is impossible to solve. Let v_1, v_2, …, v_n be nodes. At the beginning of round 0, consider k many one length paths as follows: $P_1(= v_1 \sim v_2)$, $P_2(= v_3 \sim v_4)$, …, $P_{k-1}(= v_{n-3} \sim v_{n-2})$ and $P_k(= v_{n-1} \sim v_n)$. Let $\alpha_r(w)$ represent the number of agents located at a node w at the beginning of round r, and

[1] If exploration is impossible then so is perpetual exploration and if perpetual exploration is possible then so is exploration.

let $\beta_r(w)$ represent the number of agents located at the node w at the end of round r.[2] Let $\alpha_0(v_i) = n - i - 1$ for $i \in [1, n-2]$, and $\alpha_0(v_{n-1}) = \alpha_0(v_n) = 0$ (i.e., nodes v_{n-1} and v_n are holes). Let's denote this configuration by $\mathcal{C}_0$ (refer to Fig. 1). The total number of agents is $\sum_{i=1}^{n} \alpha_0(v_i) = \sum_{i=1}^{n-2} i = \frac{(n-2)(n-1)}{2}$. In round $r \geq 0$, the adversary maintains $\mathcal{G}_r$ as follows:

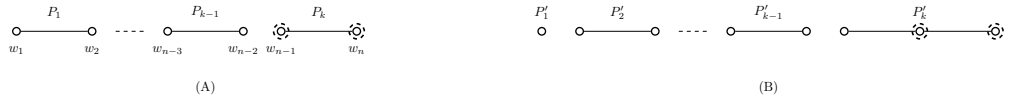


Figure 2 (A) Graph $\mathcal{G}_{iT-2}$, (B) Graph $\mathcal{G}_{iT-1}$.



Figure 3 (A) An agent moves from node w_{n-2} to node w_{n-1} at round $r-1$ in $\mathcal{G}_{r-1}$, (B) $\mathcal{G}_r$ with respect to $\mathcal{G}_{r-1}$.

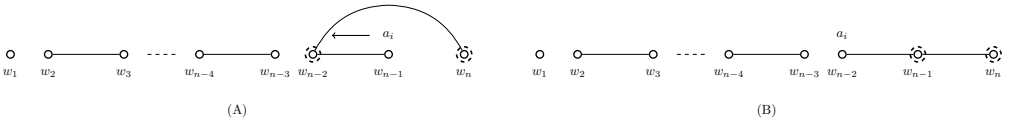


Figure 4 (A) An agent moves from node w_{n-1} to node w_{n-2} at round $r-1$ in $\mathcal{G}_{r-1}$, (B) $\mathcal{G}_r$ with respect to $\mathcal{G}_{r-1}$.

(a) $\boldsymbol{r \in [0, T-2]}$ **:** It maintains $\mathcal{C}_0$ in these rounds.

(b) $\boldsymbol{r \in [iT-1, (i+1)T-2]}$**, where** $\boldsymbol{i \geq 1 \,\&\, i \pmod 2 \neq 0}$:

At round $iT-2$, there are k many one length paths, say $P_1(= w_1 \sim w_2)$, $P_2(= w_3 \sim w_4)$, $\ldots$, $P_{k-1}(= w_{n-3} \sim w_{n-2})$, $P_k(= w_{n-1} \sim w_n)$ (we separately show that nodes w_{n-1} and w_n are holes at round $iT-2$, with $w_n = v_n$). Note that at the end of round $T-2$, $w_i = v_i$, for every i. We can see $\mathcal{G}_{iT-2}$ in Fig. 2(A). Based on the movement of agents at round $iT-2$, the adversary forms k paths, say P'_1, P'_2, $\ldots$, P'_k, at the beginning of round $iT-1$ as follows.

If $\beta_{iT-2}(w_1) \geq \beta_{iT-2}(w_2)$, then $P'_1 = w_1$. Else, $P'_1 = w_2$. For $j \in [2, k-1]$, P'_j is defined as follows.

$$P'_j = \begin{cases} w_{2j-3} \sim w_{2j-1}, & \text{if } \beta_{iT-2}(w_{2j-3}) < \beta_{iT-2}(w_{2j-2}) \text{ and } \beta_{iT-2}(w_{2j-1}) \geq \beta_{iT-2}(w_{2j}) \\ w_{2j-2} \sim w_{2j-1}, & \text{if } \beta_{iT-2}(w_{2j-3}) \geq \beta_{iT-2}(w_{2j-2}) \text{ and } \beta_{iT-2}(w_{2j-1}) \geq \beta_{iT-2}(w_{2j}) \\ w_{2j-3} \sim w_{2j}, & \text{if } \beta_{iT-2}(w_{2j-3}) < \beta_{iT-2}(w_{2j-2}) \text{ and } \beta_{iT-2}(w_{2j-1}) < \beta_{iT-2}(w_{2j}) \\ w_{2j-2} \sim w_{2j}, & \text{if } \beta_{iT-2}(w_{2j-3}) \geq \beta_{iT-2}(w_{2j-2}) \text{ and } \beta_{iT-2}(w_{2j-1}) < \beta_{iT-2}(w_{2j}) \end{cases}$$

If $\beta_{iT-2}(w_{n-3}) \geq \beta_{iT-2}(w_{n-2})$, then $P'_k = w_{n-2} \sim w_{n-1} \sim w_n$. Else, $P'_k = w_{n-3} \sim w_{n-1} \sim w_n$. We can see $\mathcal{G}_{iT-1}$ in Fig. 2(B).

Without loss of generality, let $P'_1(= w_1)$, $P'_2(= w_2 \sim w_3)$, $\ldots$, $P'_{k-1}(= w_{n-4} \sim w_{n-3})$, $P'_k(= w_{n-2} \sim w_{n-1} \sim w_n)$. If $\alpha_{iT-1}(w_{n-2}) = 0$, the adversary maintains the graph $\mathcal{G}_r$ as

[2] The parameter $\beta_r(w)$ is the number of agents at node w after completing CCM cycle at round r.

$\mathcal{G}_{iT-1}$ for every $r \in [iT, (i+1)T-2]$. If $\alpha_{iT-1}(w_{n-2}) > 0$, the adversary maintains the graph $\mathcal{G}_r$ for every $r \in [iT, (i+1)T-2]$ as follows.

If an agent from node w_{n-2} moves to node w_{n-1} at round $r-1$, then adversary at the beginning of round r maintains the following path: $P'_1 = w_1$, $P'_2 = w_2 \sim w_3$,..., $P'_{k-1} = w_{n-4} \sim w_{n-3}$, and $P'_k = w_{n-1} \sim w_{n-2} \sim w_n$ (refer Fig. 3(A) at round $r-1$, and refer Fig. 3(B) at round r). Otherwise, it maintains the graph $\mathcal{G}_r$ as $\mathcal{G}_{r-1}$.

If an agent from node w_{n-1} moves to node w_{n-2} at round $r-1$, then adversary at the beginning of round r maintains the following path: $P'_1 = w_1$, $P'_2 = w_2 \sim w_3$,..., $P'_{k-1} = w_{n-4} \sim w_{n-3}$, and $P'_k = w_{n-2} \sim w_{n-1} \sim w_n$ (refer Fig. 4(A) at round $r-1$, and refer Fig. 4(B) at round r). Otherwise, it maintains the graph $\mathcal{G}_r$ as $\mathcal{G}_{r-1}$.

If $i = 1$, then it is not difficult to observe that the number of agents at node $\alpha_{T-1}(w_{n-2}) \leq 1$ as $\alpha_{T-2}(x_{n-3}) + \alpha_{T-2}(x_{n-2}) = 3$. We show later for every i, $\alpha_{iT-1}(w_{n-2}) \leq 1$. Thus, the way we change the graph, the agent always stays between node w_{n-2} and w_{n-1} and can not access node w_n in round r. We denote this configuration by $\mathcal{C}_{1-2-3}$.

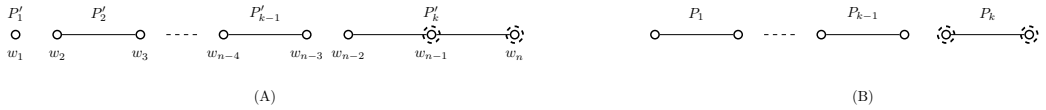


Figure 5 (A) Graph $\mathcal{G}_{iT-2}$, (B) Graph $\mathcal{G}_{iT-1}$.

(c) $r \in [iT-1, (i+1)T-2]$, where $i \geq 1$ & $i \pmod 2 = 0$:
At the end of round $iT-2$, there are k many paths, say $P'_1(= w_1)$, $P'_2(= w_2 \sim w_3)$, ..., $P'_{k-1}(= w_{n-4} \sim w_{n-3})$, $P'_k = w_{n-2} \sim w_{n-1} \sim w_n$ (we separately show that nodes w_{n-1} and w_n are holes at the beginning of round $iT-2$, with $w_n = v_n$, and at most one agent occupy node w_{n-2}). We can see $\mathcal{G}_{iT-2}$ in Fig. 5(A). At round $iT-1$, the adversary forms k many one length paths, say P_1, P_2, ..., P_k. Based on the movement of agents at round $iT-2$, the adversary forms k paths, say P_1, P_2, ..., P_k, at the beginning of round $iT-1$ as follows.

If $\beta_{iT-2}(w_2) \geq \beta_{iT-2}(w_3)$, then $P_1 = w_1 \sim w_2$. Else, $P_1 = w_1 \sim w_3$. For $j \in [2, k-2]$, P_j is defined as follows.

$$P_j = \begin{cases} w_{2j-2} \sim w_{2j}, & \text{if } \beta_{iT-2}(w_{2j-2}) < \beta_{iT-2}(w_{2j-1}) \text{ and } \beta_{iT-2}(w_{2j}) \geq \beta_{iT-2}(w_{2j+1}) \\ w_{2j-1} \sim w_{2j}, & \text{if } \beta_{iT-2}(w_{2j-2}) \geq \beta_{iT-2}(w_{2j-1}) \text{ and } \beta_{iT-2}(w_{2j}) \geq \beta_{iT-2}(w_{2j+1}) \\ w_{2j-2} \sim w_{2j+1}, & \text{if } \beta_{iT-2}(w_{2j-2}) < \beta_{iT-2}(w_{2j-1}) \text{ and } \beta_{iT-2}(w_{2j}) < \beta_{iT-2}(w_{2j+1}) \\ w_{2j-1} \sim w_{2j+1}, & \text{if } \beta_{iT-2}(w_{2j-2}) \geq \beta_{iT-2}(w_{2j-1}) \text{ and } \beta_{iT-2}(w_{2j}) < \beta_{iT-2}(w_{2j+1}) \end{cases}$$

At the end of round $iT-2$, there is at most one agent in path P'_k, and this agent is either at node w_{n-2} or w_{n-1} (we show this separately). If there is no agent in path P'_k, then P_{k-1} and P_k are defined as follows. If $\beta_{iT-2}(w_{n-4}) \geq \beta_{iT-2}(w_{n-3})$, then $P_{k-1} = w_{n-3} \sim w_{n-2}$ and $P_k = w_{n-1} \sim w_n$. Else, $P_{k-1} = w_{n-4} \sim w_{n-2}$ and $P_k = w_{n-1} \sim w_n$. If there is one agent in path P'_k, then P_{k-1} and P_k are defined as follows. Based on agent's position at node w_{n-2} or w_{n-1} at the end of round $iT-2$, the cases are as follows.

- If an agent is at node w_{n-2} at the end of round $iT-2$, the path P_{k-1} and P_k are defined as follows. If $\beta_{iT-2}(w_{n-4}) \geq \beta_{iT-2}(w_{n-3})$, then $P_{k-1} = w_{n-3} \sim w_{n-2}$ and $P_k = w_{n-1} \sim w_n$. Else, $P_{k-1} = w_{n-4} \sim w_{n-2}$ and $P_k = w_{n-1} \sim w_n$.
- If an agent is at node w_{n-1} at the end of round $iT-2$, the path P_{k-1} and P_k are defined as follows. If $\beta_{iT-2}(w_{n-4}) \geq \beta_{iT-2}(w_{n-3})$, then $P_{k-1} = w_{n-3} \sim w_{n-1}$ and $P_k = w_{n-2} \sim w_n$. Else, $P_{k-1} = w_{n-4} \sim w_{n-1}$ and $P_k = w_{n-2} \sim w_n$.

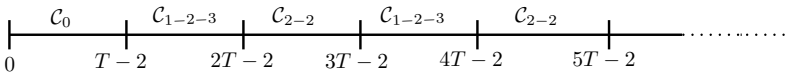


Figure 6 It shows how the agent configurations evolve periodically over the dynamic graph, with specific configurations like $\mathcal{C}_0$, $\mathcal{C}_{1-2-3}$, and $\mathcal{C}_{2-2}$ reappearing at regular intervals.

We can see $\mathcal{G}_{iT-1}$ in Fig. 5(B). It maintains $\mathcal{G}_r$ as $\mathcal{G}_{iT-1}$ for every $r \in [iT, (i+1)T-2]$. Later, we show that there is no agent in P_k, and one of the nodes in P_k is v_n. We denote this configuration by $\mathcal{C}_{2-2}$.

Fig. 6 shows how the configuration evolves over time.

▶ **Lemma 1.** *Dynamic graph G maintains the Connectivity Time property.* ***(Proof in Appendix, see Section A.1)***

We define two notations as follows.

- **(N1)** $\boldsymbol{r \in [iT-1, (i+1)T-2]}$**, where** $\boldsymbol{i \geq 1 \,\&\, i \pmod 2 = 0}$ (or $\boldsymbol{r \in [0, T-2]}$): In this case, either configuration $\mathcal{C}_0$ or $\mathcal{C}_{2-2}$ is true. Therefore, at round r, there are k one length paths P_1, P_2, ..., P_k. Let $P_j (= w_{2j-1} \sim w_{2j})$, for every $j \in [1, k]$. If $\alpha_r(w_{2j-1}) \geq \alpha_r(w_{2j})$, then we denote w_{2j-1} as w^r_{2j-1} and w_{2j} as w^r_{2j}. Otherwise, we denote w_{2j-1} as w^r_{2j} and w_{2j} as w^r_{2j-1}.
- **(N2)** $\boldsymbol{r \in [iT-1, (i+1)T-2]}$**, where** $\boldsymbol{i \geq 1 \,\&\, i \pmod 2 \neq 0}$: In this case, configuration $\mathcal{C}_{1-2-3}$ is true. Therefore, at round r, there are k paths P'_1, P'_2, ..., P'_k. Let $P'_j (= w_{2j-2} \sim w_{2j-1})$, for every $j \in [2, k-1]$, $P'_1 (= w_1)$ and $P'_k (= w_{n-2} \sim w_{n-1} \sim w_n)$. If $\alpha_r(w_{2j-2}) \geq \alpha_r(w_{2j-1})$, then we denote w_{2j-2} as w^r_{2j-2} and w_{2j-1} as w^r_{2j-1}. Otherwise, we denote w_{2j-2} as w^r_{2j-1} and w_{2j-1} as w^r_{2j-2}. We consider w_1 as w^r_1, w_{n-2} as w^r_{n-2}, w_{n-1} as w^r_{n-1}, and w_n as w^r_n.

▶ **Lemma 2.** *The following inequality holds $\forall\ l \in [1, n-2]$ and for every round $r \geq 0$.*

$$\sum_{i=1}^{l} \alpha_r(w^r_i) \geq \sum_{i=1}^{l} (n-i-1)$$

Proof. Proving our lemma is equivalent to showing that the following two inequalities hold for every $j \in [1, k-1]$ at round r:

$$\textbf{(A)} \quad \sum_{i=1}^{2j} \alpha_r(w^r_i) \geq \sum_{i=1}^{2j} (n-i-1), \qquad \textbf{(B)} \quad \sum_{i=1}^{2j-1} \alpha_r(w^r_i) \geq \sum_{i=1}^{2j-1} (n-i-1).$$

We use mathematical induction to prove inequalities (A) and (B) for every $r \geq 0$. Both inequalities are true for $r = 0$ as $\mathcal{C}_0$ includes k paths: $P_1 (= v_1 \sim v_2)$, $P_2 (= v_3 \sim v_4)$, ..., $P_{k-1} (= v_{n-3} \sim v_{n-2})$, and an additional path $P_k (= v_{n-1} \sim v_n)$. We define $\alpha_0(v_i) = n-i-1$ for $i \in [1, n-2]$, with $\alpha_0(v_{n-1}) = \alpha_0(v_n) = 0$. Therefore, $w^0_i = v_i$ for every $i \leq n-2$.

Assuming the statement is true for round $r \geq 0$, we aim to show that it also holds for round $r+1$. There are five possible cases to consider: **Case 1:** $r, r+1 \in [0, T-2]$, **Case 2:** $r, r+1 \in [iT-1, (i+1)T-2]$, where i is an odd number, **Case 3:** $r, r+1 \in [iT-1, (i+1)T-2]$, where i is an even number, **Case 4:** $r = iT-2$, where i is odd, and **Case 5:** $r = iT-2$, where i is even. Due to the length of the proof, we present only the proofs of Case 1 and Case 4 here. The proofs of Case 2 and Case 3 follow similar ideas to Case 1, and Case 5 builds upon the approach used in Case 4. The proof of Case 2, 3 and 5 is in Appendix, see Section A.2.

Case 1. In this case, agents are in configuration $\mathcal{C}_0$ at round r and $r+1$.

Proof of (A). Due to the induction hypothesis, the following inequality holds:

$$\sum_{i=1}^{2j} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j}(n-i-1) \tag{1}$$

Since the dynamic graph does not change between rounds 0 and $T-2$, no matter how agents move, for every $j \in [1, k-1]$, we have the following:

$$\alpha_r(w_{2j-1}^r) + \alpha_r(w_{2j}^r) = \alpha_{r+1}(w_{2j-1}^{r+1}) + \alpha_{r+1}(w_{2j}^{r+1}) \tag{2}$$

Therefore, the following inequality holds using Eq. (1) and Eq. (2):

$$\sum_{i=1}^{2j} \alpha_{r+1}(w_i^{r+1}) = \sum_{i=1}^{2j} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j}(n-i-1) \tag{3}$$

Proof of (B). The inequality $\alpha_r(w_1^r) \geq n-2$ holds due to the induction hypothesis, and the inequality $\alpha_r(w_1^r) + \alpha_r(w_2^r) \geq (n-2)+(n-3)$ is valid due to proof of (A). Therefore, regardless of how the agents move in round r, we have $\alpha_{r+1}(w_1^{r+1}) \geq n-2$ as $\alpha_r(w_1^r) + \alpha_r(w_2^r) = \alpha_{r+1}(w_1^{r+1}) + \alpha_{r+1}(w_2^{r+1})$ and $\alpha_{r+1}(w_1^{r+1}) \geq \alpha_{r+1}(w_2^{r+1})$. This shows (B) holds for $j=1$. For $j \in [2, k-1]$, we use a contrapositive argument. Suppose for some smallest value $j \geq 2$, the inequality (B) does not hold for round $r+1$. Then the following inequality must be true:

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) < \sum_{i=1}^{2j-1}(n-i-1) \tag{4}$$

Due to Eq. (3), we get:

$$\sum_{i=1}^{2j-2} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j-2}(n-i-1) \tag{5}$$

Therefore, using Eq. (4) and Eq. (5), the inequality $\alpha_{r+1}(w_{2j-1}^{r+1}) < n-(2j-1)-1 = n-2j$ holds. Rewrite Eq. (3) as:

$$\alpha_{r+1}(w_{2j}^{r+1}) \geq \sum_{i=1}^{2j}(n-i-1) - \sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) \tag{6}$$

From Eq. (4) and Eq. (6), the inequality $\alpha_{r+1}(w_{2j}^{r+1}) > n-2j$ holds. Thus, due to our assumption (i.e., Eq. 4), we have $\alpha_{r+1}(w_{2j-1}^{r+1}) < n-2j$ and $\alpha_{r+1}(w_{2j}^{r+1}) > n-2j$. Therefore, we have $\alpha_{r+1}(w_{2j-1}^{r+1}) < \alpha_{r+1}(w_{2j}^{r+1})$. This leads to a contradiction because, due to (N1), the inequality $\alpha_{r+1}(w_{2j-1}^{r+1}) \geq \alpha_{r+1}(w_{2j}^{r+1})$ holds. This shows that our initial assumption is incorrect. Therefore, inequality (B) holds for round $r+1$.

Case 4. In this case, at round $r = iT-2$, where i is an odd number, the configuration is either $\mathcal{C}_0$ or $\mathcal{C}_{2\text{-}2}$, and at round $r+1$, it changes to $\mathcal{C}_{1-2-3}$

Proof of (A). Due to the induction hypothesis, the following inequality is true:

$$\sum_{i=1}^{2j} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j}(n-i-1) \tag{7}$$

The adversary constructs the dynamic graph in round $r+1$ based on the agents' positions at the end of round r. Let $p = \max\{\beta_r(w^r_{2j-1}), \beta_r(w^r_{2j})\}$. Therefore, $\alpha_{r+1}(w_{2j-1}) = p$ and the value of $\alpha_{r+1}(w_{2j})$ is as follows (recall configuration $\mathcal{C}_{1-2-3}$):

$$\alpha_{r+1}(w^{r+1}_{2j}) = \max\Big\{\alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j}) - p, \max\big\{\beta_r(w^r_{2j+1}), \beta_r(w^r_{2j+2})\big\}\Big\} \tag{8}$$

The adversary forms the dynamic graph at round $iT-1$, ensuring the following equality:

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w^{r+1}_i) = \sum_{i=1}^{2j-2} \alpha_r(w^r_i) + p \tag{9}$$

Using Eq. (9), the following equality holds.

$$\sum_{i=1}^{2j} \alpha_{r+1}(w^{r+1}_i) = \sum_{i=1}^{2j-1} \alpha_{r+1}(w^{r+1}_i) + \alpha_{r+1}(w^{r+1}_{2j}) \geq \sum_{i=1}^{2j-2} \alpha_r(w^r_i) + p + \alpha_{r+1}(w^{r+1}_{2j}) \tag{10}$$

Due to Eq. (8), we have $\alpha_{r+1}(w^{r+1}_{2j}) \geq \alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j}) - p$. Thus, using Eq. (10):

$$\sum_{i=1}^{2j} \alpha_{r+1}(w^{r+1}_i) \geq \sum_{i=1}^{2j} \alpha_r(w^r_i) \tag{11}$$

Using Eq. (7) and Eq. (11), the inequality (A) holds for round $r+1$.

Proof of (B). For $j=1$, the inequality $\alpha_r(w^r_1) \geq n-2$ holds, and due to the proof of (A), the inequality $\alpha_r(w^r_1) + \alpha_r(w^r_2) \geq n-2+n-3$ holds. Therefore, $\alpha_{r+1}(w^{r+1}_1) \geq n-2$ holds regardless of how agents move at round r due to the following reason. If $\alpha_{r+1}(w^{r+1}_1) < n-2$, then $\beta_r(w^r_1) < n-2$ and $\beta_r(w^r_2) < n-2$ holds as per the dynamic graph construction at round $iT-1$, $\alpha_{r+1}(w^{r+1}_1) = \max\{\beta_r(w^r_1), \beta_r(w^r_2)\}$. Therefore, $\beta_r(w^r_1) \leq n-3$ and $\beta_r(w^r_2) \leq n-3$, and hence the inequality $\beta_r(w^r_1) + \beta_r(w^r_2) \leq 2(n-3)$ holds. Since $\beta_r(w^r_1) + \beta_r(w^r_2) = \alpha_r(w^r_1) + \alpha_r(w^r_2)$, the inequality $\alpha_r(w^r_1) + \alpha_r(w^r_2) \leq 2(n-3)$. This leads to a contradiction as $\alpha_r(w^r_1) + \alpha_r(w^r_2) \geq n-2+n-3$. Therefore, our assumption is wrong, and it implies $\alpha_{r+1}(w^{r+1}_1) \geq n-2$. Now we prove (B) when $j \geq 2$. Due to (A), the following inequalities hold for $j \geq 2$.

$$\sum_{i=1}^{2j-2} \alpha_r(w^r_i) \geq \sum_{i=1}^{2j-2} (n-i-1) \quad \text{and} \quad \sum_{i=1}^{2j} \alpha_r(w^r_i) \geq \sum_{i=1}^{2j} (n-i-1) \tag{12}$$

In Eq. (9), the lower bound of p is $\left\lceil \frac{\alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j})}{2} \right\rceil$. Using Eq. (9), we have:

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w^{r+1}_i) \geq \sum_{i=1}^{2j-2} \alpha_r(w^r_i) + \left\lceil \frac{\alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j})}{2} \right\rceil \tag{13}$$

Using Eq. (12), we have (by taking the sum of inequalities of Eq. (12)):

$$2\left(\sum_{i=1}^{2j-2} \alpha_r(w^r_i)\right) + \alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j}) \geq 2\left(\sum_{i=1}^{2j-2} (n-i-1)\right) + (n-2j) + (n-2j-1)$$

$$\implies \sum_{i=1}^{2j-2} \alpha_r(w^r_i) + \frac{\alpha_r(w^r_{2j-1}) + \alpha_r(w^r_{2j})}{2} \geq \sum_{i=1}^{2j-2} (n-i-1) + n - 2j - \frac{1}{2} \tag{14}$$

Since $\lceil x \rceil \geq x$ for every $x \in \mathbb{R}$, the following holds:

$$\sum_{i=1}^{2j-2} \alpha_r(w_i^r) + \left\lceil \frac{\alpha_r(w_{2j-1}^r) + \alpha_r(w_{2j}^r)}{2} \right\rceil \geq \sum_{i=1}^{2j-2} (n-i-1) + n - 2j = \sum_{i=1}^{2j-1} (n-i-1) \quad (15)$$

Due to Eq. (13) and (15), inequality (B) holds for round $r+1$. This completes the proof. ◀

▶ **Corollary 3.** *At round $iT-1$, $\alpha_{iT-1}(w_{n-2}^{iT-1}) \leq 1$, where i is an odd number.*

Proof. Due to Lemma 2, the following two inequalities hold at round $iT-1$.

$$\sum_{i=1}^{2n-3} w_i^{iT-1} \geq \sum_{i=1}^{2n-3} (n-i-1) \quad \text{and} \quad \sum_{i=1}^{2n-2} w_i^{iT-1} \geq \sum_{i=1}^{2n-2} (n-i-1) \quad (16)$$

Therefore, due to Eq. (16), $\alpha_{iT-1}(w_{n-2}^{iT-1}) \leq 1$. This completes the proof. ◀

▶ **Theorem 4.** *If the initial configuration contains at least two holes, then a group of $(n-2)(n-1)/2$ agents cannot solve the exploration problem in dynamic graphs that maintain the Connectivity Time property. This impossibility holds even if agents have infinite memory, full visibility, global communication, and know the parameters k, $n(\geq 4)$, $T(\geq 2)$.*

Proof. Our dynamic graph construction satisfies the Connectivity Time property as established in Lemma 1. To prove that exploration is impossible, it suffices to show that node v_n remains a hole at every round $r \geq 0$. If $r \in [0, T-2]$, node v_n is not accessible to the agents because node v_n is in path P_k, where v_{n-1} is a hole.

As per Corollary 3, $\alpha_{T-1}(w_{n-2}^{T-1}) \leq 1$. Therefore, at the beginning of round $T-1$, there can be at most one agent at node w_{n-2}. If there is no agent, then node v_n is not accessible to the agents because node v_n is in path P'_k, where w_{n-2}, w_{n-1} are holes.

If there is an agent at node w_{n-2} at the beginning of round $T-1$, it has the option to move to node w_{n-1} during round $r \in [T-1, 2T-3]$. In this scenario, the adversary constructs the following graph at round $r+1$ according to our dynamic graph construction method: the adversary maintains the paths $P'_1, P'_2, \ldots, P'_{k-1}$ unchanged and modifies P'_k from $w_{n-2} \sim w_{n-1} \sim v_n$ to $w_{n-1} \sim w_{n-2} \sim v_n$. In this manner, at round $r+1$, the node v_n is two hops away from the agent's position. The adversary follows the same procedure; if the agents move in later rounds from node w_{n-1} to w_{n-2}, this ensures that during rounds $r \in [T-1, 2T-2]$, the agents consistently remain at nodes w_{n-2} and w_{n-1}. Consequently, the agent can not reach node v_n in any of these rounds.

At the beginning of round $2T-1$, as per our dynamic graph construction, if there was an agent in path P'_k at round $2T-2$ (there is at most one agent due to Corollary 3), it removes such a node from P'_k at the beginning of round $2T-1$. Therefore, as per dynamic graph construction at the beginning of round $2T-1$, the node v_n is in path $P_k = w_{n-1} \sim w_n (= v_n)$, where node w_{n-1} is a hole. The adversary maintains the same configuration at every round $r \in [2T-1, 3T-2]$. Therefore, the agent can not reach node v_n in any of these rounds. This idea can be extended for $r \geq 3T-1$ using Corollary 3. It is important to note that this is independent of the agents' power. Therefore, the proof remains valid even if the agents have full visibility, global communication, and know all parameters. This completes the proof. ◀

6 Connectivity Time Dynamic Graph Exploration

The Connectivity Time dynamic graph has high dynamicity, and the high dynamicity may prevent even basic coordination if agents are significantly restricted in their ability to perceive their surroundings or communicate with one another. Assume agents have 1-hop visibility.

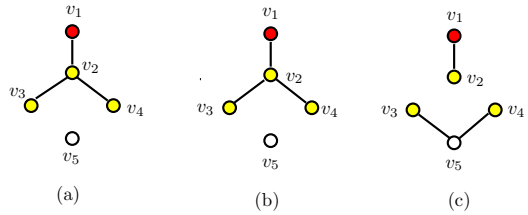


Figure 7 (a) Graph $\mathcal{G}_r$, where r(mod 3)=0, (b) Graph $\mathcal{G}_r$, where r(mod 3)=1, (c) Graph $\mathcal{G}_r$, where r(mod 3)=2. This figure is an example of the Connectivity Time for $T = 3$.

Consider two paths: $P = w_1 \sim w_2 \sim w_3 \sim w_4 \sim w_5$ and $P' = w_5 \sim w_2 \sim w_3 \sim w_4 \sim w_1$. The adversary can create either P or P', such that only at w_3, the 1-hop view remains unchanged. The movement of agent(s) at w_3 remains the same for both P and P' irrespective of the hole position. This can cause exploration to fail. Now, assume agents have global communication but no 1-hop visibility. Then, local views at each node remain the same in the above example, making it difficult again. To address these challenges, we assume that agents are equipped with 1-hop visibility and global communication capabilities in our algorithm. Even stronger assumptions are considered in the study of 1-Interval Connected dynamic networks (e.g., in [10], [35], authors use full visibility), and are instrumental in enabling tractable algorithmic solutions under highly dynamic conditions. Since 1-Interval Connectivity is a stronger assumption than the Connectivity Time, the Connectivity Time model imposes greater difficulty. It is important to emphasize that these assumptions are not fundamental to the definition of the problem but are adopted solely to overcome the adversarial nature of the Connectivity Time model. Moreover, we complement our algorithmic results with lower bounds and impossibility results (refer to Section 5) that hold even when agents possess *stronger capabilities*, including full knowledge of all system parameters, full visibility, and global communication. This highlights the inherent difficulty of perpetual exploration in such settings.

High-level idea. Suppose that at round r, agents know the map of $\mathcal{G}_r$. Let w be a multinode and v a hole in $\mathcal{G}_r$, connected via a shortest path $v_1 \sim v_2 \sim \cdots \sim v_p$, with $v_1 = w$, $v_p = v$, and all intermediate nodes $v_2, \ldots, v_{p-1}$ occupied by some agent. Let a_i denote an agent at v_i for $1 \leq i \leq p-1$. In round r, each a_i moves to v_{i+1}. The multinode v_1 remains non-empty, and each v_i $(2 \leq i \leq p-1)$ receives an agent from v_{i-1} and sends one to v_{i+1}, preserving non-hole status. Finally, the hole v_p is filled. This strategy is known as *pipeline strategy*, which pushes an agent to the nearest hole without creating new holes and has been used in prior works (e.g., [29]).

Challenges. A key challenge arises from the agents' inability to reconstruct the dynamic graph due to limited knowledge of the adversary's behavior. However, with *1-hop visibility* and *global communication*, agents can share local views to collaboratively form a partial network snapshot. While this may not capture the full graph, it often suffices for executing the pipeline procedure.

The core difficulty lies in ensuring every node is visited at least once. Even if agents can fill holes, some nodes may remain unvisited. Under the Connectivity Time model, two nodes may never belong to the same connected component in any single round. For example, let $V = \{v_1, v_2, v_3, v_4, v_5\}$ with $T = 3$, where v_1 is a multinode, and nodes v_2, v_3, v_4 are not holes; node v_5 is initially a hole. Define $\mathcal{G}_r = (V, E(r))$ as follows: if $r \bmod 3 = 0$ or 1,

let $E(r) = \{(v_1, v_2), (v_2, v_3), (v_2, v_4)\}$ (see Fig. 7(a), (b)), and if $r \bmod 3 = 2$, let $E(r) = \{(v_1, v_2), (v_5, v_3), (v_5, v_4)\}$ (see Fig. 7(c)). Although $\mathcal{G}_{r,3} = (V, \bigcup_{i=r}^{r+2} E(i))$ is connected for all r, no direct path exists between v_1 and v_5 in any single round. Thus, v_1 cannot pipeline to v_5, despite being a multinode. This illustrates how the adversary can isolate nodes round by round, impeding exploration. To address this, we use an *enhanced pipeline*. If at round r, agents detect that there exists at least one hole and at least one multinode in their connected component of $\mathcal{G}_r$, then the standard pipeline fills it. If not, then each node in the component has at least one agent. Let $\text{Count}(v)$ be the number of agents at node v at round r. If two nodes v and w in the same component satisfy $\text{Count}(w) \geq \text{Count}(v) + 2$, agents initiate a redistribution along a shortest path $v_1 \sim v_2 \sim \cdots \sim v_p$ from w to v (with $v_1 = w$, $v_p = v$), transferring one agent via pipelining to balance the load of agents. Since the graph can be highly sparse and disconnected, enhanced pipeline gradually builds a configuration of agents on the nodes such that pipeline becomes feasible along every connected path in each round.

In the following sections, we use two parameters: $\mathsf{Count}(v)$, which denotes the number of agents at node v, and $a_i.\mathsf{ID}$, which denotes the ID of agent a_i.

6.1 Map Construction

We begin by presenting an algorithm that allows agents to construct a partial map of the network, specifically, the subgraph induced by the nodes currently occupied by agents. A similar strategy was proposed in [29]; we adapt and modify it to suit our setting and terminology. This serves as a subroutine in our exploration algorithm.

At round r, since $\mathcal{G}_r$ may be disconnected, it consists of p connected components, denoted $G_1, G_2, \ldots, G_p$, where each $G_i = (V_i, E_i)$ with $V_i \subseteq V$ and $E_i \subseteq E(r)$. For each connected component G_i, we further divide it into several subgraphs considering nodes that contains agent(s). In this context, we define:

▶ **Definition 5 (Connected component of G_i without holes).** *The component G_i can be partitioned into subgraphs $G_i^1, G_i^2, \ldots, G_i^k$, where each $G_i^j = (V_i^j, E_i^j)$ satisfies the following:*

- *Every node $v \in V_i^j$ have at least one agent, for all $j \in [1, k]$.*
- *The sets of nodes and edges are pairwise disjoint, i.e., $V_i^j \cap V_i^l = \emptyset$ and $E_i^j \cap E_i^l = \emptyset$ for all $j \neq l$, where $j, l \in [1, k]$.*
- *There exists no edge $e = (u, v) \in E_i$ such that $u \in V_i^j$ and $v \in V_i^l$, where $j \neq l$, i.e., the subgraphs are disconnected from each other within G_i.*

We refer to the collection of subgraphs G_i^j as the connected component of G_i without holes, and denote by $AC(G_i)$.

In other words, $AC(G_i)$ denotes the collection of connected components obtained by removing all holes and their associated edges from G_i. Recall that agents use global communication, but communication is limited within each connected component. That is, agents located in G_i cannot communicate with agents in G_j for $i \neq j$, as communication happens through the graph links. An agent a_j in component G_i performs the following steps at round r to compute $AC(G_i)$. The algorithm is divided into two phases as described below.

Phase 1. (1-hop view collection) Let a_j be the agent with the minimum ID located at a node $v \in G_i$. The agent a_j performs the following for each port $p \in \{0, 1, \ldots, \deg_r(v) - 1\}$:

- Let u be the neighbor of v reachable via port p. Set $ID_v = a_j.\mathsf{ID}$.
- If at least one agent is present at u: define $C_v^p = (\text{Count}(v),\ ID_v,\ p,\ ID_u)$, where ID_u is the minimum ID among the agents at node u.

- If no agent is present at u (i.e., it is a hole): define $C_v^p = (\mathsf{Count}(v), ID_v, p, \perp)$, where $\perp$ denotes a hole via port p.

Let $C_v = \{C_v^0, C_v^1, \ldots, C_v^d\}$ denote the 1-hop view at node v, where $d = \deg_r(v) - 1$. The agent a_j broadcasts C_v to all agents in G_i using global communication.

Phase 2. (Graph Reconstruction) After receiving all 1-hop views from agents in G_i, agent a_j proceeds as follows:

- Define $V' = \{ID_v \mid C_v\}$, where each unique ID represents a node.
- Construct the edge set E' as follows: for each pair of tuples $(\mathsf{Count}(v), ID_v, p, ID_u)$ and $(\mathsf{Count}(u), ID_u, q, ID_v)$, where $ID_u \neq \perp$ and $ID_v \neq \perp$, add an undirected edge (ID_v, ID_u) with port labels $\pi(ID_v, ID_u) = p$ and $\pi(ID_u, ID_v) = q$, where $\pi(v_1, v_2)$ denotes the outgoing port from node v_1 to v_2.
- For each tuple $(\mathsf{Count}(v), ID_v, p, \perp)$, mark port p at node ID_v as leading to a hole.[3]

We denote this algorithm as MAP(). The correctness of MAP() is as follows. We show that at round r, if agent a_i is in connected component G_i of $\mathcal{G}_r$, then it constructs $AC(G_i)$ by using MAP(). Let G_i' be the map constructed by agent a_i using MAP().

▶ **Theorem 6.** *Let agent a_i be located at some node in the connected component G_i. Then, using the subroutine MAP(), agent a_i constructs $AC(G_i)$.* ***(Proof in Appendix, see Section A.3)***

From Theorem 6 and Definition 5, we derive three key observations.

▶ **Observation 7.** *Using MAP(), the agents in G_i construct $AC(G_i)$, including information of the number of agents at each node v and the ports from node v (if any) that lead to a hole.*

▶ **Observation 8.** *Since all agents within the same connected component exchange information via global communication, the output of MAP() is identical for every agent in that component.*

▶ **Observation 9.** *If there is no hole in G_i, then $AC(G_i)$ is the same as G_i, as per Def. 5.*

6.2 Perpetual Exploration Algorithm

In this section, we present the algorithm EXP_ALGO(), which solves perpetual exploration when agents are equipped with 1-hop visibility and global communication. The following is a detailed description of the algorithm for agent a_i at node v during round r: if node v is a multinode and agent a_i is not the minimum ID agent, then it stays at node v. If agent a_i is the minimum ID agent at node v, it follows the following steps at round r.

- Agent a_i broadcasts C_v.
- After receiving all C_us, agent a_i use $MAP()$ algorithm. Let's call this constructed graph G'. Using Theorem 6, the graph G' is nothing but a collection of partitioned subgraphs $G_j^1, G_j^2, \ldots, G_j^k$ of G_j, where G_j is the connected component for which a_i is a part. The following four cases may arise in the constructed map G', and in each case, agent a_i makes a corresponding decision.

  Case 1 (there is no multinode and no information of a port towards a hole): Agent a_i stays at node v.

[3] This step does not introduce any new holes or edges leading to holes during Phase 2; it only marks the port(s) at node v that lead to a hole.

Case 2 (there is no multinode, but there is information of a port leading to a hole): Assume there are k nodes, each with an agent that has a port leading to a hole. Let b_1, b_2, ..., b_k be agents at node u_1, u_2, ..., u_k, respectively in G', where one of the ports of u_j leads to a hole for every $j \in [1,k]$. If $a_i.ID = \min\{b_j.ID : j \in [1,k]\}$, then it moves to the node which leads to the hole via the minimum port. Otherwise, it stays at its position.

Case 3 (both a multinode and information of a port towards a hole are present): Suppose there are k multinodes in G'. Let b_1, b_2, ..., b_k be agents at node u_1, u_2, ..., u_k, respectively in G' where each node u_i is a multinode in G'. Without loss of generality, let $b_1.ID = \min\{b_j.ID : j \in [1,k]\}$. Without loss of generality, let b_1 be in G_j^1. One of the nodes in G_j^1 leads to a hole as there is a hole in G_j, and using Definition 5. Assume there are k' nodes in G_j^1, each with an agent that has a port leading to a hole. Let $\overline{a}_1$, $\overline{a}_2$, ..., $\overline{a}_{k'}$ be agents at node $\overline{u}_1$, $\overline{u}_2$, ..., $\overline{u}_{k'}$, respectively in G_j^1 such that one of the ports lead to a hole from node u_j, for every $j \in [1,k]$. Without loss of generality, let $\overline{a}_1.ID = \min\{\overline{a}_j.ID : j \in [1,k']\}$.
Since, agent a_i is aware of G', it consider a shortest path P between u_1 and $\overline{u}_1$. If there are multiple shortest paths between u_1 and $\overline{u}_1$, then it selects the one that is lexicographically shortest among all other shortest paths. Let $P = w_1(=u_1) \sim w_2 \sim \ldots \sim w_y(=\overline{u}_1)$. If agent a_i is at some node w_j, for $1 \leq j < y$, then it moves to node w_{j+1}. Else if agent a_i is at node w_y, then it moves to the node via the minimum available port, which leads to a hole. Otherwise, it stays at node v.

Case 4 (there is a multinode but there is no information of a port towards a hole): Assume there are k nodes in G'. Due to Observation 8, the graph G' is the graph G_1. Define: $i' = \max\{\mathsf{Count}(x) : x \in V_1\}$, and $j' = \min\{\mathsf{Count}(x) : x \in V_1\}$, where V_1 is the set of nodes in G_1. Let $v_1, v_2, \ldots, v_p$ be the nodes in G_1 such that $\mathsf{Count}(v_k) = i'$ for each $k \in [1,p]$, and let $\overline{a}_k$ denote the minimum ID agent at node v_k. Similarly, let $w_1, w_2, \ldots, w_q$ be the nodes in G_1 such that $\mathsf{Count}(w_{k'}) = j'$ for each $k' \in [1,q]$, and let $\overline{b}_{k'}$ denote the minimum ID agent at node $w_{k'}$. Without loss of generality, let $\overline{a}_1$ be the agent among $\{\overline{a}_k : k \in [1,p]\}$ with the minimum ID, and $\overline{b}_1$ be the agent among $\{\overline{b}_{k'} : k' \in [1,q]\}$ with the minimum ID. If $\mathsf{Count}(v_1) < \mathsf{Count}(w_1) + 2$, agent a_i stays at node v. Otherwise (i.e.,$\mathsf{Count}(v_1) \geq \mathsf{Count}(w_1) + 2$), agent a_i find a shortest path P between v_1 and w_1. If there are multiple shortest paths between v_1 and w_1, then it selects the one that is lexicographically shortest. Let $P = z_1(=v_1) \sim z_2 \ldots \sim z_y(=w_1)$. If agent a_i is at some node z_j, for $1 \leq j < y$, then it moves to node w_{j+1}. Else, it stays at v.

In the next section, we show the correctness of our algorithm.

6.3 Correctness and Analysis of the Algorithm

Before proving correctness, we introduce the following notation. Let n be the number of nodes, and define $l = \frac{(n-2)(n-1)}{2} + 1$. For each $i \in [0,l]$, let $S_i := \{v \in V : \mathsf{Count}(v) = i\}$. Let L denote the largest integer such that $S_L \neq \emptyset$ at round 0.

▶ **Lemma 10.** *Let $\mathcal{G}_r$ be the configuration at round r. If there exists at least one hole and at least one multinode in the same connected component of $\mathcal{G}_r$, then the total number of holes in $\mathcal{G}_r$ decreases by at least one at the end of round r.* ***(Proof in Appendix, see Section A.4)***

▶ **Lemma 11.** *Let $\mathcal{G}_r$ be the configuration at round r, and suppose that there exists a connected component G_1 of $\mathcal{G}_r$ such that G_1 contains at least one multinode but no hole. Define $i' = \max\{\mathsf{Count}(x) : x \in V_1\}$ and $j' = \min\{\mathsf{Count}(x) : x \in V_1)\}$, where V_1 is the*

set of nodes in G_1. If $i' \geq j'+2$, then at the end of round r, one of the nodes $v \in V(G_1)$ with Count$(v) = i'$ *reduces its* Count *by 1, and one of the nodes $w \in V(G_1)$ with* Count$(w) = j'$ *increases its* Count *by 1.* ***(Proof in Appendix, see Section A.5)***

▶ Remark 12. Based on Lemma 11, we observe that one agent reaches a node $w_1 \in S_{j'}$ at round r. As a result, Count(w_1) becomes $j'+1$ at the beginning of round $r+1$, implying that $w_1 \in S_{j'+1}$. Whenever at round r, a node becomes a part of S_p, i.e., at round $r-1$, it was not a part of S_p, we call this event *join*. Whenever at round r, a node does not remain a part of S_p, i.e., at round $r-1$, it was a part of S_p but it is not part of S_p at round r, we call this event *leave*.

We have an observation based on EXP_ALGO(), which is as follows.

▶ **Observation 13.** *A node v with* Count$(v) \geq 1$ *at round r can become a hole by the end of round r only if* Count$(v) = 1$ *and v belongs to a connected component of $\mathcal{G}_r$ that contains a hole but no multinode.*

We now show that perpetual exploration is achieved using l agents.

▶ **Lemma 14.** *If $|S_0| = 1$ at some round r, then node $v \in S_0$ is visited by some agent within the next T rounds.* ***(Proof in Appendix, see Section A.6)***

▶ **Lemma 15.** *If $|S_0| \geq 2$, then $\exists\ i, j\ (\geq i) \in [1, l]$ such that $S_i \neq \emptyset$, $S_j \neq \emptyset$, $j \geq i+2$ and $S_k = \emptyset$ for all $i < k < j$.* ***(Proof in Appendix, see Section A.7)***

▶ **Lemma 16.** *If initially $|S_0| \geq 2$, then within $O(n^4 \cdot T)$ round $|S_0| \leq 1$.*

Proof. At round 0, there are two possible cases: Case 1: $|S_1| = 0$, or Case 2: $|S_1| \geq 1$.

Case 1. To maintain the Connectivity Time, some node from S_0 must be in the same connected component as a node from S_i, $i \geq 2$, within the first T rounds. By Lemma 10, this causes $|S_0|$ to decrease by at least one in the next T rounds. If $|S_1| = 0$ throughout $[0, n \cdot T]$, then S_0 eventually becomes empty. Otherwise, if $|S_1| \geq 1$ at some round $r \in [0, n \cdot T]$, we are in Case 2.

Case 2. Suppose that at round $r \geq 0$, $|S_1| \geq 1$. To show that $|S_0| \leq 1$, we proceed by contrapositive argument. Assume that $|S_0| \geq 2$ throughout the interval $[r, r + (n^4+1) \cdot T]$. Then, by Lemma 15, there exist indices i and j with $j \geq i+2$ such that $S_i \neq \emptyset$, $S_j \neq \emptyset$, and $S_k = \emptyset$ for all $k \in [i+1, j-1]$.

According to Remark 12, nodes may join or leave the set S_p at each time step. A key question is: how many times can nodes join the set S_p? By Lemma 11, a node can join S_p at round $t \geq r$ only if at least one node from S_q (with $q \geq p+1$) and one node from S_{p-1} belong to the same connected component G_1 of $\mathcal{G}_t$, and no node from $S_{p'}$ for $p' < p-1$ is in G_1. We have l many agents. Then in the worst case, nodes can join the set S_p at most l times, as each time one agent can move from S_q to S_{p-1}. If at most l times nodes joins set S_p, the size of S_q decreases due to Lemma 11, eventually making $S_q = \emptyset$ for every $q > p$.

As there are L such sets with $L \leq l$, the total number of join events between rounds r and $r+(n^4+1)\cdot T$ can not be more than $l^2 \leq n^4$. Now divide the interval $[r, r+(n^4+1)\cdot T]$ into n^4+1 consecutive sub-intervals of length T: each sub-interval is of the form $[r+kT, r+(k+1)T-1]$ for $k = 0, 1, \ldots, n^4+1$. If at least one node leaves some set S_j during each sub-interval, then there are n^4+1 such leave events in total. Since each leave corresponds to a join, this implies there are also n^4+1 join events. But this contradicts our earlier conclusion that there can be at most n^4 join events. By the pigeonhole principle, this contradiction implies that our assumption is false. Therefore, within $O(n^4 \cdot T)$ rounds, we must have $|S_0| \leq 1$.

Since Case 2 is reached within $O(n \cdot T)$ rounds from any initial configuration, $|S_0| \leq 1$ holds within $O(n^4 \cdot T)$ rounds overall. This completes the proof. ◀

▶ **Theorem 17.** *EXP_ALGO() solves perpetual exploration in Connectivity Time dynamic graphs using* $\frac{(n-2)(n-1)}{2} + 1$ *synchronous agents equipped with 1-hop visibility, global communication, and* $O(\log n)$ *bits of memory. The agents have no knowledge of* n, T, l.

Proof. If $|S_0| \geq 2$ initially, then by Lemma 16, $|S_0| \leq 1$ within $O(n^4 \cdot T)$ rounds. Suppose that at some round $r \geq 0$, $|S_0| \leq 1$. If $S_0 = \emptyset$, then all nodes are occupied, and by Observation 13, no node becomes a hole in future rounds. Thus, perpetual exploration is achieved.

If $S_0 = \{v\}$, then by Lemma 14, node v is visited within the next T rounds. Thus, all nodes are visited by some agent within $O(n^4 \cdot T)$ rounds, and the invariant $|S_0| \leq 1$ is preserved thereafter. This process repeats indefinitely, ensuring perpetual exploration.

Each agent requires $O(\log n)$ bits to distinguish itself. Since the computation is round-local (i.e., agents do not retain information from previous rounds), $O(\log n)$ bits of memory are sufficient. This completes the proof. ◀

▶ Remark 18. Lemma 16 shows that at least $n - 1$ nodes are occupied in the first $O(n^4 \cdot T)$ rounds. Theorem 17 ensures that from this point onward, at most one hole exists and, by Lemma 14, it is revisited within each T rounds. Therefore, this also implies that each node of the network is visited by some agent in the first $O(n^4 \cdot T)$ rounds.

7 Conclusion and Future Work

In this work, we have shown that exploration is impossible in Connectivity Time dynamic graphs by $\frac{(n-2)(n-1)}{2}$ agents starting from an arbitrary initial configuration, even if agents have infinite memory, full visibility, global communication, and knowledge of all parameters. We then presented an algorithm that solves perpetual exploration using one extra agent with only 1-hop visibility, global communication, and $O(\log n)$ memory. One can study whether this extra agent helps to reduce the assumptions of 1-hop visibility and/or global communication which we require for exploration.

References

1. A. Agarwalla, J. Augustine, W. K. Moses, S. K. Madhav, and A. K. Sridhar. Deterministic dispersion of mobile robots in dynamic rings. In *ICDCN*, pages 1–4, 2018.
2. S. Albers and M. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000. doi:10.1137/S009753979732428X.
3. Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *ICALP 2008*, pages 121–132, 2008.
4. Marjorie Bournat, Ajoy K Datta, and Swan Dubois. Self-stabilizing robots in highly dynamic environments. In *SSS 2016*, pages 54–69, 2016. doi:10.1007/978-3-319-49259-9_5.
5. Marjorie Bournat, Swan Dubois, and Franck Petit. Computability of perpetual exploration in highly dynamic rings. In *ICDCS 2017*, pages 794–804, 2017. doi:10.1109/ICDCS.2017.80.
6. J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *WG*, pages 208–219, 2010.
7. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *ACM Transactions on Algorithms*, 4(4):1–18, 2008. doi:10.1145/1383369.1383373.
8. S. Das, D. Dereniowski, and C. Karousatou. Collaborative exploration of trees by energy-constrained mobile robots. *Theor. Comp. Sys.*, 62(5):1223–1240, July 2018. doi:10.1007/S00224-017-9816-3.
9. Shantanu Das. *Graph Explorations with Mobile Agents*, pages 403–422. Springer International Publishing, 2019. doi:10.1007/978-3-030-11072-7_16.

10 Shantanu Das, Nikos Giachoudis, Flaminia L. Luccio, and Euripides Markou. Broadcasting with Mobile Agents in Dynamic Networks. In *OPODIS 2020*, pages 24:1–24:16, 2021.

11 X. Deng and C.H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999. doi:10.1002/(SICI)1097-0118(199911)32:3\%3C265::AID-JGT6\%3E3.0.CO;2-8.

12 G Di Luna, Stefan Dobrev, Paola Flocchini, and Nicola Santoro. Distributed exploration of dynamic rings. *Distributed Computing*, 33:41–67, 2020. doi:10.1007/S00446-018-0339-1.

13 Y. Dieudonné and A. Pelc. Deterministic network exploration by anonymous silent agents with local traffic reports. *ACM Transactions on Algorithms*, 11(2):1–29, 2014. doi:10.1145/2594581.

14 S. Dobrev, L. Narayanan, J. Opatrny, and D. Pankratov. Exploration of high-dimensional grids by finite automata. In *ICALP*, pages 1–16, 2019.

15 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021. doi:10.1016/J.JCSS.2021.01.005.

16 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T Spooner. Two moves per time step make a difference. In *ICALP 2019*, page 141, 2019.

17 Thomas Erlebach and Jakob T Spooner. Faster exploration of degree-bounded temporal graphs. In *MFCS 2018)*, pages 36:1–36:13, 2018. doi:10.4230/LIPICS.MFCS.2018.36.

18 Paola Flocchini, Matthew Kellett, Peter C Mason, and Nicola Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50:158–184, 2012. doi:10.1007/S00224-011-9341-8.

19 Paola Flocchini, Bernard Mans, and Nicola Santoro. On the exploration of time-varying networks. *Theoretical Computer Science*, 469:53–68, 2013. doi:10.1016/J.TCS.2012.10.029.

20 P. Fraigniaud, L. Gasieniec, D. R. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi:10.1002/NET.20127.

21 P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. *Theoretical Computer Science*, 345(2–3):331–344, 2005. doi:10.1016/J.TCS.2005.07.014.

22 P. Fraigniaud, D. Ilcinkas, and A. Pelc. Impact of memory size on graph exploration capability. *Discrete Applied Mathematics*, 156(12):2310–2319, 2008. doi:10.1016/J.DAM.2007.11.001.

23 Tsuyoshi Gotoh, Paola Flocchini, Toshimitsu Masuzawa, and Nicola Santoro. Exploration of dynamic networks: Tight bounds on the number of agents. *Journal of Computer and System Sciences*, 122:1–18, 2021. doi:10.1016/J.JCSS.2021.04.003.

24 Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Group exploration of dynamic tori. In *ICDCS 2018*, pages 775–785, 2018. doi:10.1109/ICDCS.2018.00080.

25 Tsuyoshi Gotoh, Yuichi Sudo, Fukuhito Ooshita, and Toshimitsu Masuzawa. Exploration of dynamic ring networks by a single agent with the h-hops and s-time steps view. In *SSS 2019*, pages 165–177, 2019. doi:10.1007/978-3-030-34992-9_14.

26 David Ilcinkas, Ralf Klasing, and Ahmed Mouhamadou Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *SIROCCO*, pages 250–262, 2014. doi:10.1007/978-3-319-09620-9_20.

27 David Ilcinkas and Ahmed M Wade. Exploration of the t-interval-connected dynamic graphs: the case of the ring. *Theory of Computing Systems*, 62:1144–1160, 2018. doi:10.1007/S00224-017-9796-3.

28 David Ilcinkas and Ahmed Mouhamadou Wade. On the power of waiting when exploring public transportation systems. In *OPODIS 2011*, pages 451–464, 2011. doi:10.1007/978-3-642-25873-2_31.

29 Ajay D. Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, ICDCN '19, page 218–227, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3288599.3288610.

30 Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots in the global communication model. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, ICDCN '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3369740.3369775.

31 Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots on grids. In *WALCOM: Algorithms and Computation: 14th International Conference, WALCOM 2020, Singapore, Singapore, March 31 – April 2, 2020, Proceedings*, page 183–197, Berlin, Heidelberg, 2020. Springer-Verlag. doi:10.1007/978-3-030-39881-1_16.

32 F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC*, pages 513–522, New York, NY, USA, 2010.

33 O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. *Journal of Parallel and Distributed Computing*, 74(1):2016–2026, 2014. doi:10.1016/J.JPDC.2013.07.007.

34 A. Miller and U. Saha. Fast byzantine gathering with visibility in graphs. In *Algorithms for Sensor Systems*, pages 140–153, 2020.

35 William K. Moses Jr., Amanda Redlich, and Frederick Stock. Brief Announcement: Broadcast via Mobile Agents in a Dynamic Network: Interplay of Graph Properties & Agents. In *SAND 2025*, pages 17:1–17:5, 2025. doi:10.4230/LIPICS.SAND.2025.17.

36 C. Ortolf and C. Schindelhauer. Online multi-robot exploration of grid graphs with rectangular obstacles. In *SPAA 2012*, pages 27–36, New York, NY, USA, 2012.

37 P. Panaite and A. Pelc. Exploring unknown undirected graphs. *Journal of Algorithms*, 33:281–295, 1999. doi:10.1006/JAGM.1999.1043.

38 Ashish Saxena and Kaushik Mondal. Path connected dynamic graphs with a study of dispersion and exploration. *Theoretical Computer Science*, 1050:115390, 2025. doi:10.1016/J.TCS.2025.115390.

39 Claude E Shannon. Presentation of a maze-solving machine. *Claude Elwood Shannon Collected Papers*, pages 681–687, 1993.

A Appendix

A.1 Proof of Lemma 1

Proof. For $r \geq 0$, let $\mathcal{G}_r$, $\mathcal{G}_{r+1}$, ..., $\mathcal{G}_{r+T-1}$ be consecutive T sequence of graphs, where $\mathcal{G}_i = (V, E(i))$ for $i \in [r, r+T-1]$. Suppose the above dynamic graph G does not satisfy the Connectivity Time property for some round r, i.e., $G_{r,T} := (V, \cup_r^{r+T-1} E(i))$ is not connected. It is important to note that there exists a round r' between r and $r+T-1$ such that $r' = iT - 1$, for some $i \in \mathbb{N}$.

If i is **odd**, then in each round $t \in [r, iT-2]$, there are k one length paths in $\mathcal{G}_t$: $P_1(= w_1 \sim w_2)$, $P_2(= w_3 \sim w_4)$, ..., $P_{k-1}(= w_{n-3} \sim w_{n-2})$ and $P_k(= w_{n-1} \sim w_n)$. As per the dynamic graph construction at round $iT-1$, the adversary changes paths as follows: $P'_1(= w'_1)$, $P'_2(= w'_2 \sim w'_3)$, ..., $P'_{k-1}(= w'_{n-4} \sim w'_{n-3})$ and $P'_k(= w'_{n-2} \sim w'_{n-1} \sim w'_n)$, where $w'_{2j-1} \in \{w_{2j-1}, w_{2j}\}$ and $w'_{2j} = \{w_{2j-1}, w_{2j}\} \setminus \{w'_{2j-1}\}$, for every $j \in [1, k]$. Taking the union of the edges from $\mathcal{G}_i$ for $r \leq i \leq r+T-1$ creates a path of length n.

Similarly, if i is **even**, then in each round $t \in [r, iT-2]$, there are k paths in $\mathcal{G}_t$: $P'_1(= w_1)$, $P'_2(= w_2 \sim w_3)$, ..., $P'_{k-1}(= w_{n-4} \sim w_{n-3})$, and $P'_k(= w_{n-2} \sim w_{n-1} \sim w_n)$. By using a similar argument, we can show that the way we modify the construction at round $iT-1$ results in the union of edges from $\mathcal{G}_i$ for $r \leq i \leq r+T-1$, which forms a path of length n.

This shows our assumption is wrong. Therefore, this dynamic setting satisfies the Connectivity Time property. ◀

A.2 Proof of Lemma 2 (Remaining Cases)

Proof. The proof for Cases 2, 3, and 5 is as follows.

Case 2. In this case, round $r, r+1 \in [iT-1, (i+1)T-2]$, where i is an odd number, and agents are in configuration $\mathcal{C}_{1-2-3}$. The proof of Eq. (A) and Eq. (B) for round $r+1$ is as follows.

Proof of (B). Due to the induction hypothesis, the following inequality holds for $j \geq 1$:

$$\sum_{i=1}^{2j-1} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j-1} (n-i-1) \tag{1}$$

Since the dynamic graph does not change for every round $r \in [iT-1, (i+1)T-2]$, no matter how agents move, for every $j \in [1, k-2]$, we have the following

$$\alpha_r(w_{2j}^r) + \alpha_r(w_{2j+1}^r) = \alpha_{r+1}(w_{2j}^{r+1}) + \alpha_{r+1}(w_{2j+1}^{r+1}) \tag{2}$$

Therefore, inequality (B) holds for round r + 1 using Eq. (1) and Eq. (2).

Proof of (A). We use a contrapositive argument. Suppose for some smallest value $j \geq 1$, the inequality does not hold. Then the following inequality must be true:

$$\sum_{i=1}^{2j} \alpha_{r+1}(w_i^{r+1}) < \sum_{i=1}^{2j} (n-i-1) \tag{3}$$

Due to proof of (B), we have:

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j-1} (n-i-1) \tag{4}$$

Therefore, using Eq. (3) and Eq. (4), the inequality $\alpha_{r+1}(w_{2j}^{r+1}) < n-2j-1 \implies \alpha_{r+1}(w_{2j-1}^{r+1}) \leq n-2j-2$ holds. Due to proof of (B), we have:

$$\sum_{i=1}^{2j+1} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j+1} (n-i-1) \tag{5}$$

We now rewrite Eq. (5) as:

$$\alpha_{r+1}(w_{2j+1}^{r+1}) \geq \sum_{i=1}^{2j+1} (n-i-1) - \sum_{i=1}^{2j} \alpha_{r+1}(w_i^{r+1}) \tag{6}$$

From Eq. (3) and Eq. (6), the inequality $\alpha_{r+1}(w_{2j+1}^{r+1}) > n-2j-2$ holds. Therefore, due to our assumption (i.e., Eq. (3)), we have $\alpha_{r+1}(w_{2j-1}^{r+1}) \leq n-2j-2$ and $\alpha_{r+1}(w_{2j+1}^{r+1}) > n-2j-2$. This leads to a contradiction because, due to (N2), the inequality $\alpha_{r+1}(w_{2j}^{r+1}) \geq \alpha_{r+1}(w_{2j+1}^{r+1})$ holds. This shows that our initial assumption is incorrect. Therefore, inequality (A) holds for round r + 1.

Case 3. In this case, round $r, r+1 \in [iT-1, (i+1)T-2]$, where i is an even number. The proof is similar to Case 1.

Case 5. In this scenario, let $r = iT-2$, where i is an even integer. At round r, the configuration is $\mathcal{C}_{1-2-3}$. As per (N2), there are k paths: $P'_1(= w_1^r)$, $P'_2(= w_2^r \sim w_3^r)$, $\ldots$, $P'_{k-1}(= w_{n-4}^r \sim w_{n-3}^r)$, and $P'_k(= w_{n-2}^r \sim w_{n-1}^r \sim w_n^r)$. At round $r+1$ as per (N1), there are k paths: $P_1(= w_1^{r+1} \sim w_2^{r+1})$, $P_2(= w_3^{r+1} \sim w_4^{r+1})$, $\ldots$, $P_{k-1}(= w_{n-3}^{r+1} \sim w_{n-2}^{r+1})$, and $P_k(= w_{n-1}^{r+1} \sim w_n^{r+1})$. It holds that $\alpha_r(w_{2j-1}^{r+1}) \leq \alpha_r(w_{2j}^{r+1})$ for every $j \in [1, k-1]$.

Proof of (B). Due to the induction hypothesis, the following inequality is true:

$$\sum_{i=1}^{2j-1} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j-1} (n-i-1) \tag{7}$$

Since $\alpha_{r+1}(w_1^r) = \max\{\alpha_r(w_1^r), \max\{\beta_r(w_2^r), \beta_r(w_3^r)\}\}$, $\alpha_{r+1}(w_1^r) \geq \alpha_r(w_1^r) \geq n-2$. Therefore, for $j=1$, the inequality (A) holds at round $r+1$. For $j \geq 2$, the inequality (A) holds at round $r+1$ for the following reason. Let $p = \max\{\beta_r(w_{2j-2}^r), \beta_r(w_{2j-1}^r)\}$ for $j \geq 2$. Therefore, the value of $\alpha_{r+1}(w_{2j-1})$ is as follows.

$$\alpha_{r+1}(w_{2j-1}^{r+1}) = \max\Big\{\alpha_r(w_{2j-2}^r) + \alpha_r(w_{2j-1}^r) - p, \max\big\{\beta_r(w_{2j}^r), \beta_r(w_{2j+1}^r)\big\}\Big\} \tag{8}$$

The adversary forms the dynamic graph at round $iT-1$, ensuring the following equality is true.

$$\sum_{i=1}^{2j-2} \alpha_{r+1}(w_i^{r+1}) = \sum_{i=1}^{2j-3} \alpha_r(w_i^r) + p \tag{9}$$

Using Eq. (9), the following equality holds.

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) = \sum_{i=1}^{2j-2} \alpha_{r+1}(w_i^{r+1}) + \alpha_{r+1}(w_{2j-1}^{r+1}) \geq \sum_{i=1}^{2j-3} \alpha_r(w_i^r) + p + \alpha_{r+1}(w_{2j-1}^{r+1}) \tag{10}$$

Due to Eq. (8), $\alpha_{r+1}(w_{2j-1}^{r+1}) \geq \alpha_r(w_{2j-2}^r) + \alpha_r(w_{2j-1}^r) - p$. Therefore, we have the following from Eq. (10).

$$\sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j-3} \alpha_r(w_i^r) + p + \alpha_{r+1}(w_{2j-1}^{r+1}) \geq \sum_{i=1}^{2j-3} \alpha_r(w_i^r) + p + \alpha_r(w_{2j-2}^r) + \alpha_r(w_{2j-1}^r) - p$$

$$\implies \sum_{i=1}^{2j-1} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j-1} \alpha_r(w_i^r) \tag{11}$$

Using Eq. (7) and Eq. (11), the inequality (B) holds at round $r+1$.

Proof of (A). Due to the proof of (B), the following inequalities are true for any $j \geq 1$.

$$\sum_{i=1}^{2j-1} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j-1} (n-i-1) \tag{12}$$

$$\sum_{i=1}^{2j+1} \alpha_r(w_i^r) \geq \sum_{i=1}^{2j+1} (n-i-1) \tag{13}$$

Due to Eq. (9), the following inequality is true for $j \geq 1$.

$$\sum_{i=1}^{2j} \alpha_{r+1}(w_i^{r+1}) = \sum_{i=1}^{2j-1} \alpha_r(w_i^r) + p \tag{14}$$

The lower bound of p is $\left\lceil \frac{\alpha_r(w_{2j}^r) + \alpha_r(w_{2j+1}^r)}{2} \right\rceil$. Therefore, we get the following inequality from Eq. (14).

$$\sum_{i=1}^{2j} \alpha_{r+1}(w_i^{r+1}) \geq \sum_{i=1}^{2j-1} \alpha_r(w_i^r) + \left\lceil \frac{\alpha_r(w_{2j}^r) + \alpha_r(w_{2j+1}^r)}{2} \right\rceil \tag{15}$$

Using Eq. (12) and Eq. (13), we get the following inequality (by taking the sum of Eq. (12) and Eq. (13)):

$$2\left(\sum_{i=1}^{2j-1}\alpha_r(w_i^r)\right)+\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)\geq 2\left(\sum_{i=1}^{2j-1}(n-i-1)\right)+(n-2j-2)+(n-2j-1)$$

$$\implies \sum_{i=1}^{2j-1}\alpha_r(w_i^r)+\frac{\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)}{2}\geq \sum_{i=1}^{2j-1}(n-i-1)+\frac{2n-4j-3}{2}$$

$$\implies \sum_{i=1}^{2j-1}\alpha_r(w_i^r)+\frac{\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)}{2}\geq \sum_{i=1}^{2j-1}(n-i-1)+n-2j-1-\frac{1}{2} \tag{16}$$

We know that the following inequality is true.

$$\sum_{i=1}^{2j-1}\alpha_r(w_i^r)+\left\lceil\frac{\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)}{2}\right\rceil\geq \sum_{i=1}^{2j-1}\alpha_r(w_i^r)+\frac{\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)}{2} \tag{17}$$

Due to Eq. (16) and Eq. (17), the following holds.

$$\sum_{i=1}^{2j-1}\alpha_r(w_i^r)+\left\lceil\frac{\alpha_r(w_{2j}^r)+\alpha_r(w_{2j+1}^r)}{2}\right\rceil\geq \sum_{i=1}^{2j-1}(n-i-1)+n-2j-1=\sum_{i=1}^{2j}(n-i-1) \tag{18}$$

Due to Eq. (15) and Eq. (18), the following holds.

$$\sum_{i=1}^{2j}\alpha_{r+1}(w_i^{r+1})\geq\sum_{i=1}^{2j}(n-i-1)$$

This completes the proof. ◀

A.3 Proof of Theorem 6

Proof. Let edge (u,v) be in graph G_i^j, for some j, which is a subgraph of G_i, and $\pi(u,v)=p$, $\pi(v,u)=q$. Since $(u,v)\in E_i^j$, there is at least one agent at each node u and v using Definition 5. Let agent a be the minimum ID agent at node u, and agent b the minimum ID agent at node v. Since node u and v are in the same connected component, agent a gets information of C_v, and agent b gets information of C_u. Since q is an outgoing port of node v, agent a gets information about C_v^q. And since p is an outgoing port of node u, agent b gets information about C_u^p, $C_v^q=(Count(v),b.ID,q,a.ID)$ and $C_u^p=(Count(u),a.ID,p,b.ID)$. As per MAP(), agent a add edge $(a.ID,b.ID)$, where $a.ID$ and $b.ID$ are two nodes in G_i', and $\pi(a.ID,b.ID)=p$, and $\pi(b.ID,a.ID)=q$. Therefore, $G_i'=AC(G_i)$. This completes the proof. ◀

A.4 Proof of Lemma 10

Proof. Let $G_1,G_2,\ldots,G_k$ be the connected components of $\mathcal{G}_r$. Without loss of generality, suppose G_1 contains at least one hole and at least one multinode. By Theorem 6 and Observation 8, all agents in G_1 possess a common map of the anonymous copy $AC(G_1)$, denoted by $G'=\{G_1^1,G_1^2,\ldots,G_1^m\}$. Suppose there are k' multinodes in G', located at nodes $u_1,u_2,\ldots,u_{k'}$, with the minimum ID agents $b_1,b_2,\ldots,b_{k'}$ occupying them. Without loss of generality, let b_1 be the agent with the minimum ID among them, i.e., $b_1.ID=\min\{b_i.ID : i\in[1,k']\}$, and assume b_1 resides in G_1^1.

Since G_1 contains at least one hole, there exists at least one node in G_1^1 from which a port leads to a hole. Let there be k'' such nodes, denoted $\overline{u}_1, \overline{u}_2, \ldots, \overline{u}_{k''}$, with corresponding minimum-ID agents $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_{k''}$ occupying them. Let $\overline{a}_1$ be the agent with the minimum ID among them, i.e., $\overline{a}_1.ID = \min\{\overline{a}_j.ID : j \in [1, k'']\}$. Since all agents in G_1 share the same map G', they identify the same pair of nodes: u_1 (a multinode) and $\overline{u}_1$ (adjacent to a hole). Let $P = (v_1 = u_1 \sim v_2 \sim \ldots \sim v_y = \overline{u}_1)$ denote the lexicographically shortest path from u_1 to $\overline{u}_1$ in G'. This path is unique due to the deterministic selection criteria based on IDs and shared knowledge of G'.

Each agent on this path identifies its current position and acts accordingly: if an agent is at node v_i for $i < y$, it moves to v_{i+1}. The agent at $v_y = \overline{u}_1$ selects the minimum available port that leads to a hole and moves through it. Thus, a hole gets filled without creating a new hole elsewhere. Therefore, the total number of holes in $\mathcal{G}_r$ decreases by at least one at the end of round r. This completes the proof. ◀

A.5 Proof of Lemma 11

Proof. Let $G_1, G_2, \ldots, G_k$ be the connected components of $\mathcal{G}_r$. Without loss of generality, let G_1 be the connected component that contains at least one multinode but no hole. Since G_1 contains a multinode but no hole, by Theorem 6 and Observation 9, every agent in G_1 constructs the map of G_1 using the procedure MAP(). Let $v_1, v_2, \ldots, v_p$ be the nodes in G_1 such that $\mathsf{Count}(v_k) = i'$ for each $k \in [1, p]$, and let a_k denote the minimum ID agent at node v_k. Similarly, let $w_1, w_2, \ldots, w_q$ be the nodes such that $\mathsf{Count}(w_{k'}) = j'$ for each $k' \in [1, q]$, and let $b_{k'}$ denote the minimum ID agent at node $w_{k'}$. Without loss of generality, let a_1 be the agent among $\{a_k : k \in [1, p]\}$ with the minimum ID, and b_1 be the agent among $\{b_{k'} : k' \in [1, q]\}$ with the minimum ID. Since all agents share the same reconstructed map G_1, they all identify the same pair of nodes v_1 and w_1 as the nodes with maximum and minimum Count values, respectively. Let $P = (v_1 = u_1 \sim u_2 \sim \cdots \sim u_y = w_1)$ be the lexicographically shortest path from v_1 to w_1 in G_1, which is uniquely determined by the map and agent ID choices. Each agent on this path identifies its position and moves accordingly: if an agent is at node u_i for $i < y$, it moves to u_{i+1}. As a result, the value of $\mathsf{Count}(v_1)$ decreases by 1, and the value of $\mathsf{Count}(w_1)$ increases by 1. This completes the proof. ◀

A.6 Proof of Lemma 14

Proof. To maintain the Connectivity Time property, node v must be connected to some node w at round t, where $t \in [r, r + T]$. Let G_1 be the connected component of $\mathcal{G}_t$ at round t such that the node v is part of the graph G_1. If G_1 has at least one multinode, then one agent moves to node v as agents in G_1 execute EXP_ALGO() due to Lemma 10.

Otherwise, all nodes in G_1 except node v contain exactly one agent. At round t, let w_1, w_2, …, w_p be neighbours of node v in G_1, and agent a_i be at node w_i. As per EXP_ALGO(), the minimum ID agent ID among a_is moves to node v. This completes the proof. ◀

A.7 Proof of Lemma 15

Proof. Suppose the lemma does not hold. It implies $|S_i| \geq 1$ for every $i \in [1, L]$, where L is the largest index satisfying $S_L \neq \emptyset$. The value L is $\leq n - 2$. Assume $L > n - 2$. Without loss of generality, let $L = n - 1$. Since $|S_i| \geq 1$ for every $i \in [1, L]$, $\sum_{i=1}^{L} |S_i| \geq n - 1$. Therefore, the total number of nodes is $|S_0| + \sum_{i=1}^{L} |S_i| \geq 2 + n - 1 = n + 1$ as $|S_0| \geq 2$ and $\sum_{i=1}^{L} |S_i| \geq n - 1$. This leads to the contradiction as n many nodes are present. Therefore, $L \leq n - 2$.

Since $|S_i| \geq 1$ for every $i \in [1, L]$, $L \leq \cup_{i=1}^{L} |S_i| \leq n-2$. Therefore, $L \leq n-2$. Define $X = \sum_{i=1}^{L} i \cdot |S_i|$. The value X denotes the total number of agents when $\mathcal{C}$ is false. The maximum value of X occurs when $|S_i| = 1$ for $1 \leq i \leq L-1$, and $|S_L| = n - |S_0| - (L-1) \leq n - L - 1$ as $|S_0| \geq 2$. Thus, $X \leq 1+2+\cdots+(L-1)+L\cdot(n-L-1) = L\cdot(L-1)/2 + L\cdot(n-L-1)$. After simplifying, we get that $X \leq L \cdot (2n - L - 3)/2 \leq (n-2)(n-1)/2$. This leads to the contradiction as the number of agents in the system is $\frac{(n-2)(n-1)}{2} + 1$. Thus, our initial assumption must be incorrect. This completes the proof. ◀

Kudzu: Fast and Simple High-Throughput BFT

Victor Shoup
Offchain Labs, New York City, NY, USA

Jakub Sliwinski
Anza, San Diego, CA, USA

Yann Vonlanthen
ETH Zürich, Switzerland

Abstract

We present KUDZU, a high-throughput atomic broadcast protocol with an integrated fast path. Our contribution is based on the combination of two lines of work. Firstly, our protocol achieves finality in just two rounds of communication if all but p out of $n = 3f + 2p + 1$ participating replicas behave correctly, where f is the number of Byzantine faults that are tolerated. Due to the seamless integration of the fast path, even in the presence of more than p faults, our protocol maintains state-of-the-art characteristics. Secondly, our protocol utilizes the bandwidth of participating replicas in a balanced way, alleviating the bottleneck at the leader, and thus enabling high throughput. This is achieved by disseminating blocks using erasure codes. Despite combining a novel set of advantages, KUDZU is remarkably simple: intricacies such as "progress certificates", complex view changes, and speculative execution are avoided.

2012 ACM Subject Classification Computer systems organization → Distributed architectures; Security and privacy → Distributed systems security

Keywords and phrases Consensus, Blockchain, Byzantine Fault Tolerance, Fast Path, State Machine Replication

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.42

Related Version *Full Version*: `https://arxiv.org/abs/2505.08771` [35]

1 Introduction

Recent years have seen a remarkable surge in popularity and development of resilient distributed systems. The area of blockchain has become a hotbed of research, where systems akin to decentralized world-computers [11, 39, 5] compete to introduce ever-improving protocols. At the heart of any such system is a *atomic broadcast* protocol [12], which allows all replicas in the network to agree on a stream of transactions.

The crucial requirement that these protocols must satisfy is *Byzantine fault tolerance (BFT)* [29], which is the ability of a system composed of n different replicas to continue to function even if some of the replicas fail in arbitrary (potentially adversarial) ways. Moreover, the network connecting the replicas can be unreliable, or even controlled by an attacker. These harsh conditions meant that early global decentralized systems suffered severe disadvantages compared to centralized counterparts, hindering adoption. Despite the challenging setting, research has continued to improve once infamously slow decentralized protocols, and decentralized systems are closing the performance gap.

One key dimension of atomic broadcast performance is the *finalization latency* of new transactions. Historically, the protocols with the best finalization latency are protocols, such as PBFT [14], where a designated leader proposes a block of transactions, and which work in the partially synchronous communication model [19]. In this model liveness is only guaranteed during periods of time where the network is well behaved, but correctness (i.e., safety) is guaranteed unconditionally. For such protocols, it is natural to measure finalization


© Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen;
licensed under Creative Commons License CC-BY 4.0



latency as the amount of time that may elapse between when the leader proposes a block and when all other replicas finalize that block. In measuring finalization latency, we assume the leader is honest and the network is well behaved. Such protocols assume $n \geq 3f + 1$, where f is a bound on the number of corrupt replicas, and achieve finalization latency as low as 3δ, where δ is the longest actual message latency between the replicas of the system. However, even 2δ finalization latency is possible in situations where no more than p replicas fail and $n \geq 3f + 2p + 1$ for $p \geq 0$ (or even $n \geq 3f + 2p - 1$ for $p \geq 1$). Protocols achieving 2δ finalization latency in these circumstances employ a special *fast path*. This type of protocol was first explored in [32]. Ideally, such protocols would still maintain a finalization latency of 3δ, even if more than p replicas fail, by running a traditional 3δ slow path alongside the fast path. However, not all fast path protocols enjoy this property (in particular, the protocol in [32] does not). Moreover, fast path protocols typically suffer from added complexity, such as complex view-change logic, "progress certificate" messages, or speculative execution logic. This added complexity has led to a history of errors (indeed, as pointed out in [1], the protocol in [32] has a liveness bug).

Another crucial dimension of atomic broadcast performance is its *throughput*, that is, the number of transactions that the protocol can process over time given the fixed bandwidth available at every replica. Unfortunatlely, as has been observed and reported in several works [37, 33, 16], leader-based protocols often suffer from a severe "bandwidth bottleneck" at the leader. However, this leader bottleneck can be easily eliminated while still maintaining the leader-based structure and all of its practical advantages [34]. This is done by using erasure codes to ensure that the leader can disseminate large blocks with *low* and *well-balanced* communication complexity.

Another desirable feature of leader-based protocols is lightweight view-change logic that suppports frequent leader rotation. In older protocols, such as PBFT, a leader is generally kept in place for an extended period of time, and a complex and somewhat inefficient view-change subprotocol is used to switch to a new leader if the current leader is suspected of being faulty by other replicas. A newer breed of protocols, typified by HotStuff [41], employ extremely lightweight view-change logic that supports frequent leader rotation. Frequently rotating leaders can be beneficial for multiple reasons, for instance, to increase fairness when block production comes with rewards (e.g., maximal extractable value), or to increase censorship resistance.

In this work, our aim is to unite and improve the state-of-the-art in these key dimensions, and to do so while keeping the protocol as simple as possible.

Our contribution. We present KUDZU[1]: a fast and high-throughput atomic broadcast protocol that is remarkably simple compared to its predecessors. KUDZU is the first BFT protocol that combines an optimistic 2δ latency fast path integrated into a 3δ latency slow path, with high-throughput data dispersal, as well as lightweight view change logic that supports frequent leader rotation. We provide a detailed description of the protocol and rigorously prove its security.

1. **Fast path.** In a network of $n = 3f + 2p + 1$ replicas, KUDZU achieves finalization latency of 2δ if at most p replicas are corrupt.
2. **Simultaneous slow path.** If more than p replicas fail, KUDZU maintains the best possible finalization latency of 3δ by running a slow path alongside the fast path.

[1] Kudzu is an insidious, fast-growing vine, also known as Mile-a-Minute.

3. **High throughput.** High throughput is achieved by KUDZU using erasure codes to ensure that the leader can disseminate large blocks with low and well-balanced communication complexity.
4. **Lightweight view change.** KUDZU employs an extremely simple and efficient view change logic that allows for frequent leader rotation.

KUDZU is the first atomic broadcast protocol to satisfy all of these properties simultaneously. It also enjoys other properties, such as optimistic responsiveness (the protocol proceeds as fast as the network will allow, with no artificially introduced delays), and a block time (the delay between successive honest leaders proposing a block) of just 2δ.

Technical Intuition. Inspired by DispersedSimplex [34], KUDZU introduces the minimal changes to that protocol required to incorporate a fast path. A designated leader in KUDZU distributes the block data by sending erasure coded fragments to all other replicas. In turn, these replicas then broadcast the fragments themselves, together with a first-round vote on the cryptographic hash identifying the block. The voting logic carefully incorporates a rule that makes sure that a block can already be finalized given $n - p$ first-round votes. As a result, replicas can already reassemble the block and finalize it in 2δ latency in the good case where the network is well behaved, the leader is honest, and at most p replicas are corrupt. Another vote-counting rule is used to finalize the block on a slower, 3δ-latency path, if more than p replicas are corrupt. To avoid getting stuck altogether, a replica may also vote for a couple of other blocks than the one it initially received from the leader, including a special "timeout" block. The main innovation of our paper is the logic for determining when these "extra" votes are cast – this logic is deceptively simple, but carefully maintains a very delicate balance between liveness and safety. With this innovation, we avoid intricacies such as "progress certificates", complex view changes, and speculative execution as found in other protocols (such as Banyan [38], SBFT [22], Kuznetsov et al. [28], and HotStuff-1 [23]).

2 Related Work

2.1 Fast Path Protocols

A long line of work proposes consensus protocols with a fast path, typically called "fast", "early-stopping" or "one-step" consensus [9, 27, 18, 36, 32, 21]. FaB Paxos [32] introduces a parametrized model with $3f + 2p + 1$ replicas, where $p \geq 0$. The parameter p describes the number of replicas that are not needed for the fast path. These protocols can terminate optimally fast in theory (2δ, or 2 network delays) under optimistic conditions. The papers [18, 28, 2] point out that the lower bound of $3f + 2p + 1$ actually only applies to a restricted type of protocol. The papers present single-shot consensus protocols that use only $3f + 2p^* - 1$ replicas, with $p^* \geq 1$, and prove the corresponding lower bound.

2.2 Simultaneous Slow Path Protocols

Interestingly, in practice, these fast path protocols might *increase* the finalization latency, as the fast path requires a round of voting between $n - p$ replicas, which could be slower than two rounds of voting between $n - f - p$ replicas that are concentrated in a geographic area. Banyan [38] performs the fast path in parallel with the 3δ mechanism, which is optimally fast if more than p replicas are faulty or exhibit higher network latency. However, Banyan can exhibit *unbounded* message complexity when there is a corrupt leader. KUDZU addresses this drawback, shares the same optimistic latency properties, improves throughput through balanced dispersal, benefits from a simpler design, and better properties during leader rotation (see Section 2.4).

2.3 High-Throughput Protocols

Leader-based protocols such as PBFT [14], HotStuff [41] and Tendermint [10] suffer from a bandwidth bottleneck, as in these protocols the leader is responsible for disseminating transactions to all replicas. As mentioned above, this bottleneck has been well known for quite a while [37, 33, 16]. One way to alleviate this bottleneck is to move away from leader-based protocols to a more symmetric, leaderless protocol design where all replicas disseminate transactions. Such an approach was already taken in [37, 33, 24, 16]. These protocols typically have worse latency than leader-based protocols, and moreover, since many replicas may end up broadcasting the same transactions, the supposed improvement in throughput can end up being illusory (note that [37] actually tackles this duplication problem head on).

Kudzu follows a different approach to address the leader bottleneck, which makes use of erasure coding [39, 20]. By splitting blocks into smaller, erasure-coded shares, the leader can transmit less data, leading to a balanced utilization of resources. This line of work shines by providing high throughput and low latency [34, 30]. Alpenglow [25] is a recent proof-of-stake protocol being deployed for the Solana blockchain that showcases the appeal of this line of work in practice. Alpenglow and Kudzu are intimately related in their voting logic.

2.4 View Changes

Some atomic broadcast protocols are notorious for having complex view changes, especially in the presence of a fast path [4]. In the case of Zzyzzva [26] and UpRight [15], safety errors were later pointed out [1]. Arguably SBFT [22] was the first to correct these mistakes. SBFT [22] avoids the all-to-all broadcast that occurs in Kudzu, but this results in higher finalization latency. Moreover, SBFT does not address the above-mentioned leader bottleneck.

3 Model and Preliminaries

We consider a network of n replicas $P_1, \ldots, P_n$ called replicas. Up to f replicas can be Byzantine, i.e., deviate from the protocol in arbitrary ways, such as collude to attack the protocol. The remaining replicas follow the protocol and are referred to as honest. We aim to provide better latency if only up to p replicas do not cooperate. In other words, $n - p$ honest replicas including the leader are enough for the fast path to be effective.

To prove the security of Kudzu we assume

$$n \geq 3f + 2p + 1, \tag{1}$$

where $f \geq 1$ and $p \geq 0$. Moreover, to prove concrete bounds on message, communication, and storage complexity, we assume

$$n < 3(f + p + 1). \tag{2}$$

Assumption (2) is not a real restriction. Indeed, if $n \geq 3(f + p + 1) = 3f + 3p + 3$, then we can always *increase* p appropriately, leaving n and f alone, so that both (1) and (2) are satisfied. This only increases the overall performance of the protocol.

3.1 Network Assumptions

We will not generally assume network synchrony. However, we say the network is δ-*synchronous at time* T if every message sent from an honest replica P at or before time T to an honest replica Q is received by Q before time $T + \delta$. We also say the the network is **δ-synchronous over an interval** $[a, b + \delta]$ if it is δ-synchronous at time T for all $T \in [a, b]$.

While our protocols always guarantee *safety*, even in periods of asynchrony, *liveness* will only be guaranteed in periods of δ-synchrony, for appropriately bounded δ (and this synchrony bound may be explicitly used by the protocol). Thus, we are essentially working in the *partial synchrony* model of [19]. However, instead of assuming a single point in time (GST) after which the network is assumed to be synchronous, we take a somewhat more general point of view that models a network that may alternate between periods of asychrony and synchrony.

Replicas have local clocks that can measure the passage of local time. We do not assume that clocks are synchronized in any way. However, we do assume that there is no *clock skew*, that is, all clocks tick at the same rate (but we could also just assume the skew is bounded and incorporate that bound into the protocol's synchrony bound).

3.2 Problem Statement

The purpose of state machine replication is to totally order blocks containing transactions, so that all replicas output transactions in the same order. Our protocol orders blocks by associating them with natural numbered slots. Some leader replica is assigned to every slot. For every slot, either some block produced by the leader might be finalized, or the protocol can yield an empty block. The guarantees of our protocol can be stated as follows:

- **Safety**: If some honest replica finalizes block B in slot v, and another honest replica finalizes block B' in slot v, then $B = B'$.
- **Liveness**: If the network is in a period of synchrony, each honest replica continues to finalize blocks for slots $v = 1, 2, \ldots$

In addition to liveness and safety, we support a fast path:

- **Fast Termination**: If the network is in a period of synchrony, $n - p$ replicas behave momentarily honestly, and an honest leader proposes a block B at time t, then every honest replica finalizes B at time $t + 2\delta$.

3.3 Cryptographic Assumptions

3.3.1 Signatures and certificates

We make standard cryptographic assumptions of secure digital signatures and collision-resistant hash functions. We assume all replicas know the public keys of other replicas.

We use a k-out-of-n threshold signature scheme. We refer to a *signature share* and a *signature certificate*: signature shares from k replicas on a given message may be combined to form a signature certificate on that message. This can be implemented in various ways, e.g., based on BLS signatures [8, 7, 6]. The security property for such a threshold signature scheme may be stated as follows.

Quorum Size Property: It is infeasible to produce a signature certificate on a message m, unless $k - f'$ honest replicas have issued signature shares on m, where $f' \leq f$ is the number of corrupt replicas.

For ease of exposition and analysis, we assume *static corruptions*, so the adversary must choose some number $f' \leq f$ replicas to corrupt at the very beginning of the protocol execution, and then does not corrupt any replicas thereafter. That said, we believe all of our the protocols are secure against *adaptive corrupts*, provided the threshold signature scheme is as well.

As we will see, we will need a one k-out-of-n threshold signature scheme with $k = n - f - p$, and another with $k = n - p$.

3.3.2 Information Dispersal

We make use of well-known techniques for *asynchronous verifiable information dispersal (AVID)* involving erasure codes and Merkle trees [13].

Erasure codes. For integer parameters $k \geq d \geq 1$, a (k, d)-*erasure code* encodes a bit string M as a vector of k *fragments*, $f_1, \ldots, f_k$, in such a way that any d such fragments may be used to efficiently reconstruct M. Note that for variable-length M, the reconstruction algorithm also takes as input the length β of M. The reconstruction algorithm may fail (for example, a formatting error) – if it fails it returns $\perp$, while if it succeeds it returns a message that when re-encoded will yield k fragments that agree with the original subset of d fragments. We assume that all fragments have the same size, which is determined as a function of k, d, and β.

Using a Reed-Solomon code, which is based on polynomial interpolation, we can realize a (k, d)-erasure code so that if $|M| = \beta$, then each fragment has size $\approx \beta/d$. In our protocol, the payload of a block will be encoded using an $(n, f + p + 1)$-erasure code. Such an erasure code encodes a payload M as a vector of fragments $f_1, \ldots, f_n$, any $f + p + 1$ of which can be used to reconstruct M. This leads to a data expansion rate of at most roughly 3; that is, $\sum_i |f_i| \approx n/(f + p + 1) \cdot |M| < 3|M|$, where last inequality follows from assumption (2).

Merkle trees. Recall that a Merkle tree allows one replica P to commit to a vector of values $(v_1, \ldots, v_k)$ using a collision-resistant hash function by building a (full) binary tree whose leaves are the hashes of $v_1, \ldots, v_k$, and where each internal node of the tree is the hash of its two children. The root r of the tree is the commitment. Replica P may "open" the commitment at a position $i \in [k]$ by revealing v_i along with a "validation path" π_i, which consists of the siblings of all nodes along the path in the tree from the hash of v_i to the root r. We call π_i a *validation path from the root under r to the value v_i at position i*. Such a validation path is checked by recomputing the nodes along the corresponding path in the tree, and verifying that the recomputed root is equal to the given commitment r. The collision resistance of the hash function ensures that P cannot open the commitment to two different values at a given position.

Encoding and decoding. For a given payload M of length β, we will encode M as a vector of fragments $(f_1, \ldots, f_n)$ using an $(n, f + p + 1)$-erasure code, and then form a Merkle tree with root r whose leaves are the hashes of $f_1, \ldots, f_n$. We define the *tag* $\tau \coloneqq (\beta, r)$.

For a tag $\tau = (\beta, r)$, we shall call (f_i, π_i) a *certified fragment for τ at position i* if

- f_i has the correct length of a fragment for a message of length β, and
- π_i is a correct validation path from the root under r to the fragment f_i at position i.

The function *Encode* takes as input a payload M. It builds a Merkle tree for M as above with root r (encoding M as a vector of fragments, and then building the Merkle tree whose leaves are the hashes of all of these fragments). It returns

$$\big(\ \tau,\ \{(f_i, \pi_i)\}_{i \in [n]}\ \big),$$

where τ is the tag (β, r), β is the length of M, and each (f_i, π_i) is a certified fragment for τ at position i.

The function *Decode* takes as input

$$\big(\ \tau,\ \{(f_i, \pi_i)\}_{i \in \mathcal{I}}\ \big),$$

where $\tau = (\beta, r)$ is a tag, $\mathcal{I}$ is a subset of $[n]$ of size $f + p + 1$, and each (f_i, π_i) is a certified fragment for τ at position i. It first reconstructs a message M' from the fragments $\{f_i\}_{i \in \mathcal{I}}$,

using the size parameter β. If $M' = \perp$, it returns $\perp$. Otherwise, it encodes M' as a vector of fragments $(f'_1, \ldots, f'_n)$ and Merkle tree with root r' from $(f'_1, \ldots, f'_n)$. If $r' \neq r$, it returns $\perp$. Otherwise, it returns M'.

Under collision resistance for the hash function used for the Merkle trees, any $f + p + 1$ certified fragments for given tag τ will decode to the same payload – moreover, if τ is the output of the encoding function, these fragments will decode to M (and therefore, if the decoding function outputs $\perp$, we can be sure that τ was maliciously constructed). This observation is the basis for the protocols in [17, 31, 40]. Moreover, with this approach, we do not need to use anything like an "erasure code proof system" (as in [3]), which would add significant computational complexity (and in particular, the erasure coding would have to be done using parameters compatible with the proof system, which would likely lead to much less efficient encoding and decoding algorithms).

4 Kudzu Protocol

Kudzu iterates through slots, where in each slot there is a designated leader who proposes a new block, which is chained to a parent block. Leaders may be rotated in each slot, either in a round-robin fashion or using some pseudo-random sequence. The slot leader disseminates large blocks in a way that keeps the overall communication complexity low and avoids a bandwidth bottleneck at the leader. The communication is balanced, meaning that each replica, including the leader, transmits roughly the same amount of data over the network. We describe our protocol as a few simple subprotocols that run concurrently with each other:

- Vote and Certificate Pool: data structure managing the votes and certificates;
- Complete Block Tree: data structure storing the reconstructed blocks;
- Main Loop: loop issuing votes that makes sure some blocks become *notarized* and *finalized*. Notarized blocks can be reconstructed by all replicas and are added to the Complete Block Tree. Finalized blocks are ordered and output by the protocol.

4.1 Protocol Data Objects

▶ **Definition 1** (block). *A* ***block*** B *is of the form* $\mathsf{Block}(v, \tau, h_p)$*, where*

(i) $v \in \{1, 2, \ldots\}$ *is the slot number associated with the block (and we say* B is a block for slot v*),*

(ii) τ *is the tag obtained by encoding* B*'s payload* M*,*

(iii) h_p *is the hash of* B*'s parent block (or* $h_p = \perp$ *by convention if* B*'s parent is a notional "genesis" block).*

We also call a certified fragment for the tag τ *a* certified fragment for B*. The block* $B_v^{\mathsf{timeout}} = \mathsf{Block}(v, \perp, \perp)$ *is a special* timeout block*.*

▶ **Definition 2** (votes and certificates). *A* ***notarization vote from*** P_i ***for block*** B *is an object of the form* $\mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i)$*, where* σ_i *is a valid signature share from* P_i *on the object* $\mathsf{Notar}(B)$*, and* (f_i, π_i) *is either a certified fragment for* B *at position* i*, or* $(f_i, \pi_i) = (\perp, \perp)$ *if* $B = B_v^{\mathsf{timeout}}$*.*

A ***notarization certificate for*** B *is an object of the form* $\mathsf{NotarCert}(B, \sigma)$*, where* σ *is a valid* $(n - f - p)$*-out-of-*n *signature certificate on the object* $\mathsf{Notar}(B)$*. The notarization vote on the timeout block is also called the timeout vote, and the notarization certificate for the timeout block is called the timeout certificate.*

*A **first vote from** P_i **on block** B is an object of the form* $\mathsf{FirstVote}(\sigma_i', \mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i))$*, where σ_i' is a valid signature share from P_i on the object* $\mathsf{First}(B)$*, and* $\mathsf{NotarVote}(B, \sigma_i, f_i, \pi_i)$ *is a notarization vote from P_i on block B.*

*A **fast finalization certificate for** $B \neq B_v^{\mathbf{timeout}}$ is an object of the form* $\mathsf{FirstCert}(B, \sigma)$*, where σ is a valid $(n-p)$-out-of-n signature certificate on the object* $\mathsf{First}(B)$*.*

*A **finalization vote from** P_i **on block** B is an object of the form* $\mathsf{FinalVote}(B, \sigma_i)$*, where σ_i is a valid signature share from P_i on the object* $\mathsf{Final}(B)$*.*

*A **finalization certificate for** B is an object of the form* $\mathsf{FinalCert}(B, \sigma)$*, where σ is a valid $(n-f-p)$-out-of-n signature certificate on the object* $\mathsf{Final}(B)$*.*

4.2 Vote and Certificate Pool

Each replica maintains a *pool* with votes and certificates. For every slot, the pool stores votes and certificates associated with the slot.

As we will see, by design, for any one slot, a honest replica can send only 1 first vote, 1 timeout vote, and 1 finalization vote. As we will also see later (in Section 5.3), for one slot a honest replica can only send at most 3 (non-timeout) notarization votes. Any votes exceeding this bound can only result from misbehavior and are not added to the pool. For example, if a replica receives more than three notarization votes for a given slot from some replica P, the replica can ignore these votes and conclude that P is corrupt. In particular, only one first vote per replica can be observed by the protocol loop in Protocol 1. When a first vote is added to the pool, also the contained notarization vote is added to the pool.

Whenever a replica receives enough votes, and it does not already have a corresponding certificate, it will generate the certificate, add it to the pool, and broadcast the certificate to all replicas. Similarly, whenever a replica receives a certificate, and it does not already have a corresponding certificate, it will add it to the pool, and broadcast the certificate to all replicas. For one slot it is impossible that the pool would receive or create more than: 1 timeout certificate, 1 fast finalization certificate, 1 finalization certificate, and 5 notarization certificates. The first bound is immediate, since there can be only one timeout certificate per slot. The second and third bounds follow from the safety analysis below. The fourth bound follows from the analysis in Section 5.3.

4.3 Complete Block Tree

Each replica also maintains a *complete block tree*, which is a tree of blocks rooted at a notional genesis block at slot 0. We will show that the number of blocks for a given slot is bounded by 5. A block $B = \mathsf{Block}(v, \tau, h_p)$ is added to the tree if each of the following holds:

- the certificate pool contains a notarization certificate for B and $B \neq B_v^{\mathsf{timeout}}$;
- $h_p = \perp$ or the complete block tree contains a parent block with the hash h_p;
- the replica has received enough (i.e. $f + p + 1$) notarization votes to reconstruct the effective payload M of B as

  $$M \leftarrow \mathit{Decode}(\tau, \{(f_i, \pi_i)\}_{i \in \mathcal{I}}),$$

  where $\{(f_i, \pi_i)\}_{i \in \mathcal{I}}$ is the corresponding collection of certified fragments for τ;
- $M \neq \perp$ and satisfies some correctness predicate that may depend on the path of blocks (and their payloads) from genesis to block B.

A replica does not broadcast anything in addition to adding a block to the block tree.

4.4 Finalization

We say that a block B for slot v is *explicitly finalized by replica* P if the complete block tree of P contains B and the certificate pool of P contains either a fast finalization certificate for B or finalization certificate for B. In this case, we say that all of the predecessors of block B in the complete block tree are *implicitly finalized by* P. The payloads of finalized blocks may be then transmitted in order to the execution layer of the protocol stack of a replicated state machine.

4.5 Generating Block Proposals

The logic for generating block proposal material B, $(f_1, \pi_1), \ldots, (f_n, \pi_n)$ in slot v in line 16 of Protocol 1 is as follows:

- build a payload M that validly extends the path in the complete block tree ending at a block B_{p} with hash h_p;
- compute

$$(\tau, \{(f_i, \pi_i)\}_{i \in [n]}) \leftarrow \mathit{Encode}(M);$$

- set $B \coloneqq \mathsf{Block}(v, \tau, h_p)$.

4.6 Validating Block Proposals

To check if $\mathsf{BlockProp}(B, f_j, \pi_j)$ is a valid block proposal in slot v in line 18 of Protocol 1, replica P_j checks that the following conditions holds:

- the proposal is signed or otherwise authenticated by the leader,
- B is of the form $\mathsf{Block}(v, \tau, h_p)$,
- (f_j, π_j) is a certified fragment for τ at position j,
- the complete block tree contains a block with the hash h_p in a slot $v' < v$;
- the pool contains timeout certificates for slots $v' + 1, \ldots, v - 1$;

These last two conditions might not hold at a given point in time, but may hold at a later point in time, and so might need to be checked again when blocks or certificates are added.

4.7 Main Loop

The main protocol for P_j is described in Protocol 1. In the description, $\mathsf{leader}(v)$ denotes the leader for slot v – as mentioned, leaders may be rotated in each slot, either in a round-robin fashion or using some pseudo-random sequence.

As mentioned in Section 4.2, each replica only considers only one first vote that it receives from any other replica. To make this explicit in the protocol, we use a map *firstVote* from replicas to blocks to record these votes for the current slot. The protocol also uses simple helper functions on this map.

- $\mathsf{allVotes}(\mathit{firstVote})$: the total number of first votes for slot v contained in the pool,
- $\mathsf{maxVotes}(\mathit{firstVote})$: the maximal number of first votes on some non-timeout block B for slot v contained in the pool,
- $\mathsf{manyVotes}(\mathit{firstVote})$: returns the set of non-timeout blocks in slot v on which the pool contains at least $f + p + 1$ first votes.

For example, if the pool contains 1, 2, 3, 4 first votes on blocks B_1, B_2, B_3, B_v^{timeout} respectively (and $f + p + 1 = 2$), then $\mathsf{allVotes} = 10$, $\mathsf{maxVotes} = 3$, and $\mathsf{manyVotes} = \{B_2, B_3\}$.

The protocol also uses a subprotocol $\mathsf{ReconstructAndNotarize}(v, B)$, defined in Protocol 2.

Each replica P_j moves through slots $v = 1, 2, \ldots$. In each slot, it will enter a loop in which it waits for one of several conditions to trigger an action. These conditions are based on the objects in its pool and its complete block tree, as well as local variables.

- Lines 9–11 present the logic for the replica successfully exiting the slot by finding a block B for that slot in its complete block tree. In addition, if the replica did not broadcast a notarization vote for any other block (including the timeout block) for that slot, it will also broadcast a finalization block for B.
- Lines 12–13 present the logic for the replica unsuccessfully exiting the slot by obtaining a timeout certificate for that slot.
- Lines 14–17 present the logic for the replica proposing a block for that slot if it is the leader for that slot. It generates the block proposal as in Section 4.5, extending the path in the complete block tree ending at B_{p}. Here, B_{p} is either the genesis block or the block that it found in its complete block tree the last time it successfully exited a slot (other choices of B_{p} are possible).
- Lines 18–21 present the logic for the replica broadcasting a first vote for a non-timeout block B. It will do so only if B is a valid block proposed by the leader (as in Section 4.6) and has not already first voted. Recall that a first vote for B also includes a notarization vote for B.
- Lines 22–25 present the logic for the replica broadcasting a first vote for the timeout block for this slot. It will do so only if a sufficient amount of time has passed since it entered the slot and has not already first voted.
- Lines 26–27 present the logic for updating the map *firstVotes*. No other actions are taken.
- Lines 28–31 present the logic for the replica taking a "second look" at a block B, if it has received sufficiently many first votes for B. It will do so only if it has already first voted (and has not already taken a second look at B). If it can reconstruct a valid payload for B, it will broadcast a notarization vote for B (if it has not already done so); otherwise, it will broadcast a notarization vote for the timeout block.
- Lines 32–35 present the logic for the replica broadcasting a timeout vote under special circumstances. It will do so only if it has already first voted and

  $$\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) \geq f + p + 1.$$

  We note that the quantity

  $$\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote})$$

  cannot decrease as we add entries to *firstVote*. That is because, when we add an entry, the first term increases by 1 and the second either decreases by 1 or remains unchanged.

5 Protocol Analysis

We start by proving some helpful properties.

▶ **Lemma 3** (Validity Property). *Suppose that a block B for some slot v is added to the complete block tree of some replica. If the leader for slot v is honest, B must have been proposed by that leader.*

Proof. By the Quorum Size Property (see Section 3.3.1) for notarization certificates, at least $n - 2f - p$ honest replicas must have broadcast notarization shares for B. Since we are assuming $n \geq 3f + 2p + 1$, it follows that $n - 2f - p > 0$, so some honest replica P must

Algorithm 1 KUDZU main loop for replica P_j.

$B_{\mathrm{p}} \leftarrow \mathsf{genesis}$ ▷ *parent of the next block*
for $v = 1, 2, \ldots$ **do**
  $T_{\mathrm{start}} \leftarrow \mathsf{clock}()$ ▷ *slot-local initialisation*
  $\mathit{done}, \mathit{proposed}, \mathit{firstVoted} \leftarrow \mathsf{false}$
  $\mathit{notarized} \leftarrow \{\}$ ▷ *blocks already notarized*
  $\mathit{secondLook} \leftarrow \{\}$ ▷ *blocks already reconsidered*
  $\mathit{firstVote} \leftarrow \{\}$ ▷ *map $P_i \mapsto B$ for their first vote*

  while $\neg \mathit{done}$ **wait until either**
    there exists a block B for slot v in the complete block tree $\Rightarrow$
      $B_{\mathrm{p}} \leftarrow B$; $\mathit{done} \leftarrow \mathsf{true}$
      if $\mathit{notarized} \subseteq \{B\}$ **then** broadcast $\mathsf{FinalVote}(B, \sigma_j)$

    the pool contains a timeout certificate for v $\Rightarrow$
      $\mathit{done} \leftarrow \mathsf{true}$

    $\neg \mathit{proposed} \wedge \mathsf{leader}(v) = P_j \Rightarrow$
      $\mathit{proposed} \leftarrow \mathsf{true}$
      generate block proposal material B, $(f_1, \pi_1), \ldots, (f_n, \pi_n)$ extending block B_{p}
      for all $i \in [n]$: **send** $\mathsf{BlockProp}(B, f_i, \pi_i)$ to P_i

    $\neg \mathit{firstVoted} \wedge$ received valid $\mathsf{BlockProp}(B, f_j, \pi_j)$ from $\mathsf{leader}(v) \Rightarrow$
      $\mathit{firstVoted} \leftarrow \mathsf{true}$
      broadcast $\mathsf{FirstVote}(\sigma'_j, \mathsf{NotarVote}(B, \sigma_j, f_j, \pi_j))$
      $\mathit{notarized} \leftarrow \mathit{notarized} \cup \{B\}$

    $\neg \mathit{firstVoted} \wedge \mathsf{clock}() > T_{\mathrm{start}} + \Delta_{\mathrm{timeout}} \Rightarrow$
      $\mathit{firstVoted} \leftarrow \mathsf{true}$
      broadcast $\mathsf{FirstVote}(\sigma'_j, \mathsf{NotarVote}(B_v^{\mathsf{timeout}}, \sigma_j, \bot, \bot))$
      $\mathit{notarized} \leftarrow \mathit{notarized} \cup \{B_v^{\mathsf{timeout}}\}$

    received valid $\mathsf{FirstVote}(_, \mathsf{NotarVote}(B, _))$ from P_i **and** $\mathit{firstVote}[P_i] = \bot \Rightarrow$
      $\mathit{firstVote}[P_i] \leftarrow B$

    $\mathit{firstVoted} \wedge \exists B \in \mathsf{manyVotes}(\mathit{firstVote}) \setminus \mathit{secondLook}$
     and B's parent is in the complete block tree $\Rightarrow$
      $\mathit{secondLook} \leftarrow \mathit{secondLook} \cup \{B\}$
      $\mathsf{ReconstructAndNotarize}(v, B)$

    $\mathit{firstVoted} \wedge (\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) \geq f + p + 1)$
     and $B_v^{\mathsf{timeout}} \notin \mathit{notarized} \Rightarrow$
      broadcast $\mathsf{NotarVote}(B_v^{\mathsf{timeout}}, \sigma_j, \bot, \bot)$
      $\mathit{notarized} \leftarrow \mathit{notarized} \cup \{B_v^{\mathsf{timeout}}\}$

Algorithm 2 ReconstructAndNotarize(v, B).

```
1: reconstruct payload for B
2: if reconstruction succeeds ∧ payload valid then
3:     if B ∉ notarized then
4:         broadcast NotarVote(B, σ_j, f_j, π_j)
5:         notarized ← notarized ∪ {B}
6: else
7:     if B_v^timeout ∉ notarized then
8:         broadcast NotarVote(B_v^timeout, σ_j, ⊥, ⊥)
9:         notarized ← notarized ∪ {B_v^timeout}
```

have broadcast a notarization share for B. This could happen either at line 20 or at line 31. In the first case, B must be the block that P received as a proposal from the leader. In the second case, since P received $f + p + 1$ first votes for B, one of these must be a first vote for B from some honest replica Q, and so B must be the block that Q received as a proposal from the leader. ◀

▶ **Lemma 4** (Completeness Property for Certificates). *If a certificate X appears in the vote and certificate pool (so X is a notarization, finalization, or timeout certificate) then X (or its equivalent) will eventually appear in the corresponding pool of every other replica. Moreover, if X appears in a replica's pool at a time T at which the network is δ-synchronous, it will appear in every replica's pool before time $T + \delta$.*

Proof. This is clear, since a certificate appearing in the vote and certificate pool is broadcast immediately. ◀

▶ **Lemma 5** (Completeness Property for Blocks). *If a block B appears in the complete block tree, then B will eventually appear in the corresponding tree of every other replica. Moreover, if B appears in a replica's tree at a time T at which the network is δ-synchronous, it will appear in every replica's tree before time $T + \delta$.*

Proof. We are relying on the Quorum Size Property (see Section 3.3.1) for notarization certificates: when a notarization certificate for a block B is added to the certificate pool, at least $n - 2f - p$ honest replicas must have already broadcast notarization votes for B, which contain B as well as fragments sufficient to reconstruct B's payload, since $n - 2f - p \geq f + p + 1$. ◀

5.1 Safety

▶ **Lemma 6** (Fast Finalization Implication). *Suppose a block B is fast finalized by some honest replica, then the number of honest replicas that first vote for anything other than B is at most p.*

Proof. Let $f' \leq f$ be the actual number of corrupt replicas. If some honest replica fast finalizes B, then – by the Quorum Size Property (see Section 3.3.1) for fast finalization certificates – at least $n - p - f'$ honest replicas first voted for B. So the number of honest replicas that first vote for anything other than B is at most $(n - f') - (n - p - f') = p$. ◀

▶ **Lemma 7** (Uniqueness of Fast Finalization Property). *If an honest replica receives $f + p + 1$ first votes for a block B in slot v, then no block different from B can be fast finalized in slot v by any honest replica.*

Proof. Suppose towards contradiction that and some replica P receives $f+p+1$ first votes for a block B but a block $C \neq B$ is fast finalized by some honest replica Q. By the previous lemma, at most p honest replicas could first vote for anything other than C. Therefore, P can receive at most $f+p$ first votes for B, a contradiction. ◀

▶ **Lemma 8** (Absence of Fast Finalization Property). *If the inequality* allVotes(*firstVote*) − maxVotes(*firstVote*) $\geq f+p+1$ *holds for a replica in slot v, then no block can be fast finalized in slot v by any replica.*

Proof. Suppose towards contradiction that a replica P fast finalized block B while for replica Q the inequality holds. By Lemma 6, at most p honest replicas fast vote for anything other than B. Therefore, Q can receive at most $f+p$ fast votes for anything other than B. Let $count_B$ denote the number of first votes for B received by Q. It follows that, at any point in time, for replica Q, we have

$$\mathsf{allVotes}(firstVote) - \mathsf{maxVotes}(firstVote) \leq \mathsf{allVotes}(firstVote) - count_B \leq f+p,$$

a contradiction. ◀

▶ **Lemma 9** (Incompatibility of Notarization and (Fast) Finalization Property). *Suppose that a valid block B for some slot v is (fast) finalized by some replica. If any replica obtains a notarization certificate for a block C in slot v, then $C = B$. (In particular, no other block for slot v can be added to the complete block tree of any replica and no timeout certificate can be obtained for slot v.)*

The proof of Lemma 9 is deferred to the appendix.

We can now easily state and prove our main safety lemma:

▶ **Lemma 10** (Safety). *Suppose a replica P explicitly finalizes a block B for slot v, and a block C for slot $w \geq v$ is in the complete block tree of some replica Q. Then B is an ancestor of C in Q's complete block tree.*

Proof. By the Incompatibility of Notarization and (Fast) Finalization Property (Lemma 9), no timeout certificate for slot v can be produced. Let C' be the parent of C and suppose w' is the slot number of C'. Since C' is in Q's complete block tree, a notarization certificate for C' must have been produced, which means at least one honest replica must have issued a notarization vote for C', which means $v \leq w' < w$. The inequality $v \leq w'$ follows from the fact that there is no timeout certificate for slot v, and an honest replica will issue a notarization share for C only if it has timeout certificates for slots $w'+1, \cdots, w-1$. If $v = w'$, we are done by the Incompatibility of Notarization and (Fast) Finalization Property (Lemma 9), and if $v < w'$, we can repeat the argument inductively with C' in place of C. ◀

5.2 Liveness

Liveness follows from the following lemmas. The first lemma analyzes the optimistic case where the network is synchronous and the leader of a given slot is honest, showing that the leader's block will be committed.

▶ **Lemma 11** (Liveness I). *Consider a slot $v \geq 1$ and suppose the leader for slot v is an honest replica Q. Suppose that the first honest replica P to enter the loop iteration for slot v does so at time T_0. Suppose that the network is δ-synchronous over the interval $[T_0, T_0+4\delta]$ for some δ with $\Delta_{\text{timeout}} \geq 2\delta$. Then, Q will propose a block for slot v by time $T \leq T_0+\delta$.*

Each honest replica will finish the loop iteration before time $T + 2\delta$ by adding Q's proposed block B to its complete block tree. Moreover, if $n - p$ replicas are honest, each honest replica will finalize B by time $T + 2\delta$. If more than p replicas are corrupt, each honest replica will finalize B by time $T + 3\delta$.

Proof. By the Completeness Properties (Lemma 4 and Lemma 5), before time $T_0 + \delta$, each honest replica will enter slot v by time $T_0 + \delta$, having either a timeout certificate for slot $v - 1$ or a block for slot $v - 1$ in its complete block tree. Before time $T \leq T_0 + \delta$, the leader Q will propose a block B that extends a block B' with slot number $v' < v$. By the logic of the protocol, we know that Q must have timeout certificates for slots $v' + 1, \ldots, v - 1$ at the time it makes its proposal, as well as a notarization certificate for B'. Again by the Completeness Properties, before time $T + \delta$, each honest replica will have B' in its complete block tree and all of these timeout certificates in its certificate pool. Each honest replica will receive B before this time, and because $\Delta_{\text{timeout}} \geq 2\delta$, will broadcast a first vote for B by this time. Because all honest replicas broadcast a first vote for B, each such replica will only see ever see at most f first votes for any other block. It follows that each honest replica will only ever see

- $\mathsf{manyVotes}(\mathit{firstVote}) \subseteq \{B\}$, and
- $\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) \leq \mathsf{allVotes}(\mathit{firstVote}) - \mathit{count}_B \leq f$, where count_B is the number of of first votes for B that it sees.

Therefore, honest replicas will not broadcast a notarization votes in slot v for anything other than B. Before time $T + 2\delta$, each honest replica will have added B to its complete block tree and broadcast a finalization vote on B. If $n - p$ replicas are honest, each honest replica will have added a fast finalization certificate to its pool by time $T + 2\delta$ as well. Otherwise, if more than p replicas are corrupt, each honest replica will finalize B before time $T + 3\delta$, when adding the finalization certificate for slot v to its pool. ◀

The second lemma analyzes the pessimistic case, when the network is asynchronous or the leader of a given slot is corrupt. It says that eventually, all honest replicas will move on to the next slot.

▶ **Lemma 12** (Liveness II). *Suppose that the network is δ-synchronous over an interval $[T, T + \Delta_{\text{timeout}} + 3\delta]$, for an arbitrary value of δ, and that at time T, some honest replica is in the loop iteration for slot v and all other honest replicas are in a loop iteration for v or a previous slot. Then, before time $T + \Delta_{\text{timeout}} + 3\delta$, all honest replicas exit slot v.*

Proof. By the Completeness Properties (Lemma 4 and Lemma 5), every honest replica will enter the slot v before time $T + \delta$. By time $T + \delta + \Delta_{\text{timeout}}$, every honest replica will broadcast a first vote either for a block proposal, or for B_v^{timeout}.

Consider two cases:

(a) At least $f + p + 1$ honest replicas broadcast a first vote for the same non-timeout block B.

(b) No set of $f + p + 1$ honest replicas broadcast a first vote for the same non-timeout block B.

Case (a). Since $f + p + 1$ replicas cast notarization votes for B, some honest replica did so. Since this replica had to have B's parent in its complete block tree, by Completeness Properties (Lemma 5) each honest replica will have B's parent in its complete block tree by time $T + \Delta_{\text{timeout}} + 2\delta$. All honest replicas observe all first votes from other honest replicas before time $T + \Delta_{\text{timeout}} + 2\delta$, and so each honest replica will call $\mathsf{ReconstructAndNotarize}(v, B)$ before that time, unless it has already exited slot v. If some honest replica has exited slot v

before that time, the by Completeness Properties (Lemma 4 and Lemma 5), all honest replicas will exit the slot before time $T + \Delta_{\text{timeout}} + 3\delta$. Otherwise, assume no honest replica has exited before time $T + \Delta_{\text{timeout}} + 2\delta$. Whenever $\mathsf{ReconstructAndNotarize}(v, B)$ is called by some honest replica, it has received at least $f + p + 1$ first votes for B, and so can attempt to reconstruct B. If it fails to reconstruct B's payload or finds that it is invalid, then it and all honest replicas will do so and issue a timeout vote (this follows from collision resistance of the hash function and the fragment decoding logic). Otherwise, each honest replica will issue a notarization vote for B. Therefore, before time $T + \Delta_{\text{timeout}} + 3\delta$, each honest replica will either add B to its complete block tree or the timeout certificate to its pool, and proceed exit the slot.

Case (b). Consider some honest replica P. By time $T + \Delta_{\text{timeout}} + 2\delta$, P will have observed all votes of other honest replicas. If the only entries in *firstVote* are those from honest replicas, then the inequality $\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) \geq f + p + 1$ must hold. To see this, if $f' \leq f$ is the number of corrupt replicas, then

$$\begin{aligned} \mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote}) &\geq (n - f') - (f + p) \\ &\geq n - 2f - p \\ &\geq f + p + 1 \quad (\text{since } n \geq 3f + 2p + 1). \end{aligned}$$

As we have already observed, the quantity $\mathsf{allVotes}(\mathit{firstVote}) - \mathsf{maxVotes}(\mathit{firstVote})$ cannot decrease as we add entries to *firstVote*. Therefore if P has not cast a timeout vote in slot v yet, it will do so by time $T + \Delta_{\text{timeout}} + 2\delta$, unless it has already exited slot v by that time. In either case, all honest replicas will exit slot v by time $T + \Delta_{\text{timeout}} + 3\delta$. ◀

5.3 Boundedness

We prove some simple results that allow us to bound message and storage complexity. Here, we are assuming both (1) and (2).

Let us consider notarization votes on non-timeout blocks. An honest replica sends a first vote for at most one such block. Any notarization vote for some other non-timeout block B requires that the replica found $B \in \mathsf{manyVotes}(\mathit{firstVotes})$. In other words, the replica has seen at least $f + p + 1$ other replica's first votes for B. At most one first vote per replica is considered when computing $\mathsf{manyVotes}(\mathit{firstVotes})$. Therefore, an honest replica can cast at most $\lfloor n/(f+p+1) \rfloor$ notarization votes beyond the first vote, and by (2), $\lfloor n/(f+p+1) \rfloor \leq 2$, for a total of 3 notarization votes for non-timeout blocks.

Suppose there are $f' \leq f$ corrupt replicas, and so $n - f'$ honest replicas. The honest replicas therefore issue at most $3(n - f')$ notarization votes for non-timeout blocks per slot.

Now, to construct a notarization certificate for a non-timeout block B, we require that $(n - f - p - f')$ honest replicas cast a notarization vote for B. Therefore, by the result in the previous paragraph, there can be at most

$$N \coloneqq \lfloor (3(n - f'))/(n - f - p - f') \rfloor \tag{3}$$

distinct blocks for which a notarization certificate can be constructed.

We claim that $N \leq 5$. To see this, first note that the derivative of $(3(n-f'))/(n-f-p-f')$ with respect to f' is positive, and therefore the right-hand side of (3) is maximized when $f = f'$, and so

$$N \leq \lfloor (3(n - f))/(n - 2f - p) \rfloor.$$

So it suffices to show that $(3(n-f))/(n-2f-p) < 6$. This is easily seen to follow by a simple calculation using (1).

So to summarize, we have shown that in any slot,

1. each replica casts a notarization vote for at most 2 non-timeout blocks besides its first vote, and
2. there are at most 5 distinct blocks for which a notarization certificate can be constructed.

This immediately gives us bounds on the message and storage complexity of the protocol per slot.

5.4 Complexity

Based on the concrete bounds in Section 5.3 (and the preliminary discussion in Section 4.2), it is easily seen that the message complexity per slot is $O(n^2)$. Based on the properties of erasure codes and Merkle trees discussed in Section 3.3.2, the communication complexity per slot is $O(\beta n + n^2 \log(n)\kappa + n^2\lambda)$, where β is a bound on the payload size, κ is the output length of the collision-resistant hash, and λ is a bound on the length of any signature share or certificate. Moreover, the communication is *balanced*, in that every replica, *including the leader*, transmits the same amount of data, up to a constant factor. It is also easily seen that each replica needs to store $O(\beta + n\log(n)\kappa + n\lambda)$ bits of data.

6 Protocol Variations

Our protocol can be adapted to the setting of $n \geq 3f + 2p^* - 1$, where $p^* \geq 1$ [2], with the insight that if honest replicas vote for different blocks in the same slot, the leader has to be corrupt. The liveness analysis can leverage the fact that, in this case, honest replicas will observe at least $n - f$ votes from non-leader replicas, as is done in [38]. We plan to analyze a variation of Kudzu with this adaptation in follow-up work.

A variation of DispersedSimplex [34] features segments of consecutive slots with the same leader. Such stable leader assignment is beneficial for the throughput of the protocol. The same technique can be applied to our protocol, and we plan to analyze a variation of Kudzu featuring stable leaders in follow-up work.

References

1 Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. Revisiting fast practical byzantine fault tolerance. *arXiv preprint arXiv:1712.01367*, 2017.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021. doi:10.1145/3465084.3467899.

3 Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. Cryptology ePrint Archive, Paper 2021/1500, 2021. URL: https://eprint.iacr.org/2021/1500.

4 Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems (TOCS)*, 32(4):1–45, 2015. doi:10.1145/2658994.

5 Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. Sui lutris: A blockchain combining broadcast and consensus. Technical report, Technical Report. Mysten Labs. https://sonnino. com/papers/sui-lutris. pdf, 2023.

6 Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2003. doi:10.1007/3-540-36288-6_3.

7 Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Paper 2018/483, 2018. URL: https://eprint.iacr.org/2018/483.

8 Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001. doi:10.1007/3-540-45682-1_30.

9 Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *Parallel Computing Technologies: 6th International Conference, PaCT 2001 Novosibirsk, Russia, September 3–7, 2001 Proceedings 6*, pages 42–50. Springer, 2001. doi:10.1007/3-540-44743-1_4.

10 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus, 2018. arXiv:1807.04938, http://arxiv.org/abs/1807.04938.

11 Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

12 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001. doi:10.1007/3-540-44647-8_31.

13 Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005. doi:10.1109/RELDIS.2005.9.

14 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

15 Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 277–290, 2009.

16 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. Also at arXiv:2105.11827, http://arxiv.org/abs/2105.11827. doi:10.1145/3492321.3519594.

17 S. Dolev and Z. Wang. SodsBC: Stream of distributed secrets for quantum-safe blockchain. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 247–256, Los Alamitos, CA, USA, 2020. IEEE Computer Society.

18 Partha Dutta, Rachid Guerraoui, and Marko Vukolic. The complexity of asynchronous byzantine consensus. Technical report, IC Technical Report 200499, École Polytechnique Fédérale de Lausanne, 2004.

19 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

20 Solana Foundation. Turbine–solana's block propagation protocol solves the scalability trilemma. https://solana.com/news/

turbine---solana-s-block-propagation-protocol-solves-the-scalability-trilemma, 2019.

21 Rachid Guerraoui and Marko Vukolić. Refined quorum systems. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 119–128, 2007.

22 Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019. doi:10.1109/DSN.2019.00063.

23 Dakai Kang, Suyash Gupta, Dahlia Malkhi, and Mohammad Sadoghi. Hotstuff-1: Linear consensus with one-phase speculation. *arXiv preprint arXiv:2408.04728*, 2024. doi:10.48550/arXiv.2408.04728.

24 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021. Also at arXiv:2102.08325, http://arxiv.org/abs/2102.08325. doi:10.1145/3465084.3467905.

25 Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana alpenglow consensus. https://www.anza.xyz/alpenglow-1-1, 2025.

26 Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

27 Klaus Kursawe. Optimistic byzantine agreement. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 262–267. IEEE, 2002. doi:10.1109/RELDIS.2002.1180196.

28 Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. Revisiting optimal resilience of fast byzantine consensus. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 343–353, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467924.

29 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

30 Andrew Lewis-Pye, Kartik Nayak, and Nibesh Shrestha. The pipes model for latency and throughput analysis. Cryptology ePrint Archive, Paper 2025/1116, 2025. URL: https://eprint.iacr.org/2025/1116.

31 Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In Yuval Emek and Christian Cachin, editors, *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 129–138. ACM, 2020. doi:10.1145/3382734.3405707.

32 J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006. doi:10.1109/TDSC.2006.35.

33 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 31–42. ACM, 2016. Also at https://eprint.iacr.org/2016/199. doi:10.1145/2976749.2978399.

34 Victor Shoup. Sing a Song of Simplex. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:22, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.37.

35 Victor Shoup, Jakub Sliwinski, and Yann Vonlanthen. Kudzu: Fast and Simple High-Throughput BFT. *CoRR*, 2025. doi:10.48550/arXiv.2505.08771.

36 Yee Jiun Song and Robbert van Renesse. Bosco: One-step byzantine asynchronous consensus. In *International Symposium on Distributed Computing*, pages 438–450. Springer, 2008. doi:10.1007/978-3-540-87779-0_30.

37 Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-BFT: High-throughput robust bft for decentralized networks, 2019. arXiv:1906.05552, http://arxiv.org/abs/1906.05552.

38 Yann Vonlanthen, Jakub Sliwinski, Massimo Albarello, and Roger Wattenhofer. Banyan: Fast rotating leader bft. In *Proceedings of the 25th International Middleware Conference*, pages 494–507, 2024. doi:10.1145/3652892.3700788.

39 Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf, 2018.

40 Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. Dispersedledger: High-throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022. URL: https://www.usenix.org/conference/nsdi22/presentation/yang.

41 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus in the lens of blockchain, 2018. arXiv:1803.05069, http://arxiv.org/abs/1803.05069.

A Proof of Lemma 9

Proof. We first prove the Incompatibility of Notarization and *Fast* Finalization. Suppose towards contradiction that for slot v a fast finalization certificate exists for block B and a notarization certificate exists for block C, with $C \neq B$. By the Quorum Size Property (see Section 3.3.1) for notarization certificates, this implies that an honest replica broadcast a notarization vote for C.

- On the one hand, suppose that the notarization vote for C was sent by the protocol on line 31. Due to the condition on line 28, this means that $f + p + 1$ first votes were received for a block D. Note that D is a non-timeout block. Moreover, by the logic of ReconstructAndNotarize, either C is a timeout block or $D = C$. By the Uniqueness of Fast Finalization Property (Lemma 7) we know that $D = B$. Since B is assumed to be valid, C cannot be a timeout block, so we also have $B = D = C$, a contradiction.
- On the other hand, suppose that the notarization vote for C was sent on line 34. By the Absence of Fast Finalization Property (Lemma 8), this implies that no block is fast finalized, again a contradiction.

The Incompatibility of Notarization and Finalization Property follows from a standard quorum intersection argument, based on the fact that in each slot an honest replica issues a finalization vote only for a block only if it did not send a notarization vote for a different block in that slot (see line 11). Suppose towards contradiction that for slot v a finalization certificate exists for block B and a notarization certificate exists for block C, with $C \neq B$. By the Quorum Size property (see Section 3.3.1) for finalization and notarization certificates, if $f' \leq f$ is the number of corrupt replicas, then at least $n - f - p - f'$ honest replicas broadcast finalization votes for B, and a disjoint set of at least the same number of honest replicas broadcast notarization votes for C. This implies that there are at least $2(n - f - p - f')$ distinct honest replicas. However, under the assumption that $n \geq 3f + 2p + 1$, we have $2(n - f - p - f') \geq n - f' + 1$, a contradiction. ◀

Weight Reduction in Distributed Protocols: New Algorithms and Analysis

Anatoliy Zinovyev
Boston University, MA, USA

Abstract

We study the problem of minimizing the total weight of (potentially many) participants of a distributed protocol, a necessary step when the original values are large but the scheme to be deployed scales poorly with the weights. We assume that α fraction of the original weights can be corrupted and we must output new weights with at most β adversarial fraction, for $\alpha < \beta$. This problem can be viewed from the prism of electing a small committee that does the heavy work, a powerful tool for making distributed protocols scalable. We solve the variant that requires giving parties potentially multiple seats in the committee and counting each seat towards the cost of the solution. Moreover, we focus on the "deterministic" version of the problem where the computed committee must be secure *for any* subset of parties that can be corrupted by the adversary; such a committee can be smaller than a randomly sampled one in some cases and is useful when security against adaptive corruptions is desired but parties in the sub-protocol speak multiple times.

Presented are new algorithms for the problem as well as analysis of prior work. We give two variants of the algorithm Swiper (PODC 2024), one that significantly improves the running time without sacrificing the quality of the output and the other improving the output for a reasonable increase in the running time. We prove, however, that all known algorithms, including our two variants of Swiper, have worst case approximation ratio $\Omega(n)$. To counter that, we give the first polynomial time algorithm with approximation factor $n/\log^2 n$ and also the first sub-exponential time exact algorithm, practical for some real-world inputs. Of theoretical interest is another polytime algorithm that we present, based on linear programming, whose output is no worse than an optimal solution to the problem with slightly different parameters.

We implemented and tested previous and new algorithms, comparing them on the stake distributions of popular proof-of-stake blockchains, and found that our second variant of Swiper computes solutions extremely close to the optimal, confirmed by our exact algorithm.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Cryptographic protocols; Theory of computation → Distributed algorithms

Keywords and phrases Weight reduction, distributed protocols, weighted cryptography, threshold cryptography, consensus, committee selection, adaptive corruptions, approximation algorithms, linear programming, rounding

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.43

Related Version *Full Version*: `https://eprint.iacr.org/2025/1076` [42]

Supplementary Material *Software (Source Code)*: `https://github.com/tolikzinovyev/weight-reduction/tree/f1b5ce44aab9e4809bff67c5c6bff4acf1f82cb1` [41]
archived at `swh:1:dir:a48c33feba04bc98c7013c4c88b174f5e461a392`

Funding This material is based upon work supported in part by a gift from DARPA under Agreements No. HR00112020021, HR00112020023 and HR001120C0085. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.
Anatoliy Zinovyev: DARPA under Agreements No. HR00112020021, HR00112020023 and HR001120C0085.

Acknowledgements The author thanks his advisor Leo Reyzin for help with preparing this paper and Nathan Klein, Rico Zenklusen for helpful discussions.


© Anatoliy Zinovyev;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Committee selection is an essential tool for making large-scale distributed protocols scalable. The idea is that, instead of all n parties doing the work, the protocol is run within a smaller committee, decided by the larger protocol; the participants outside the committee simply learn the result. To spread the work evenly or to achieve better security, multiple different committees can be created.

We are mainly interested in byzantine fault tolerant / cryptographic protocols where parties have some assigned weights w_i and the adversary can corrupt parties up to some α fraction of the total weight in the system. For example, in a proof-of-stake blockchain parties own some amount of money in the system (stake) and the assumption is that the adversary controls, say, at most 20% of all the stake. We would like to compute a "small" committee where the adversary controls less than some larger β fraction, say 1/3, of the total weight. Specifically, our goal is to assign each party a new non-negative integer weight t_i such that we minimize the total new weight $\sum_{i=1}^{n} t_i$. This is useful when the protocol in question doesn't scale well with the weights such as in secret sharing, threshold signatures or secure multiparty computation. Additionally, we focus on "deterministic" committee selection that must output a committee secure against *any* set of parties that the adversary is allowed to corrupt. Formally, the new weights $t_1, ..., t_n$ must satisfy

$$\forall A \subseteq [n] \text{ with } \sum_{i \in A} w_i \leq \alpha \sum_{i=1}^{n} w_i,\ \sum_{i \in A} t_i < \beta \sum_{i=1}^{n} t_i$$

(here A denotes the set of parties controlled by the adversary). This property is useful when one requires security against an adversary that performs adaptive corruptions throughout the protocol execution with full knowledge of $(t_1, ..., t_n)$ – for example, when the members of the committee need to speak multiple times and thus reveal themselves after the first message. Moreover, committees of this flavor can be smaller than traditional "randomized" committees, as explained in section 2. As demonstrated in [38], they also yield a convenient way to transform a wide range of unweighted protocols to their weighted version. We note that "deterministic" committee selection cannot be usefully instantiated in the unweighted setting since any committee would have to have $\Omega(n)$ members (otherwise, the adversary can corrupt the whole committee). In contrast, when parties have weights distributed non-uniformly, we can often compute a committee with the total new weight much smaller than n.

1.1 Prior work

The "deterministic" committee selection problem has a short history. To the best of our knowledge, the recent Swiper paper [38] by Tonkikh and Freitas was the first and only to study it in isolation. Near-simultaneously, other works [4, 23, 22] attempted to solve it in the context of their larger protocols. All four papers consider different variants of the problem although all aim to minimize the total new weight $\sum_{i=1}^{n} t_i$.

Swiper defines a problem they call weight restriction that is only slightly different from the one we consider: given $w_1, ..., w_n$ as input, output $t_1, ..., t_n \in \mathbb{Z}_{\geq 0}$ such that for all $A \subseteq [n]$ with $\sum_{i \in A} w_i < \alpha \sum_{i=1}^{n} w_i$, we have $\sum_{i \in A} t_i < \beta \sum_{i=1}^{n} t_i$. The only distinction is that the inequality containing α is non-strict in our paper.

The authors of Swiper prove that the problem admits an efficiently computable solution of cost $\sum_{i=1}^{n} t_i \leq O(n)$ (assuming α and β parameters are constant). Such a solution can be found by linearly scaling down the vector $(w_1, ..., w_n)$ and rounding each w_i to an integer

according to a simple rule. Swiper goes a step further and tries to compute a solution as small as possible by running a binary search on the scaling factor and testing the correctness of a vector $(t_1, ..., t_n)$ at each iteration. Its time complexity is $O(n^2 \log n)$.

We note that Swiper defines two more weight reduction problems. Their overview and the overview of the other works [4, 23, 22] that consider deterministic committee selection can be found in the full version of our paper [42, Appendix A].

1.2 Our contribution and paper outline

Section 2 gives background information on committee selection and how the problem we consider relates to other variants. In section 3 we formally define the problem and notation. Section 4 provides a detailed overview of the Swiper algorithm, useful for reading the following section.

Section 5 presents our new variants of Swiper called faster Swiper and extended Swiper. The first variant improves the time complexity of Swiper from $O(n^2 \log n)$ to $O(n+R^2)$ where $R \leq O(n)$ is the total weight in the output. This improves the running time by a $\log n$ factor in the worst case, but given that the cost of the output R is often much smaller than n, the improvement is much larger as demonstrated by our numerical evaluations. For low latency applications, our optimizations would be highly desired. Importantly, faster Swiper doesn't alter the quality of the output of Swiper. Our second variant extended Swiper improves the output quality by replacing the binary search with a linear search. Although the idea is simple, the algorithm has non-trivial time complexity of $O(n + R^{2.5})$. Numerical evaluations demonstrate a sizable reduction in the total new weight over Swiper and faster Swiper, since the two can get stuck in a local minimum; in fact, the cost of extended Swiper's output is extremely close to optimal in our tests. At the same time, its running time is comparable and often even smaller than that of Swiper. Both variants rely on two optimizations: 1) carefully bounding the running time in terms of the output cost R and 2) reusing dynamic programming computations.

We, however, prove in section 6 that all previously known algorithms, as well as our two Swiper variants, suffer from $\Omega(n)$ approximation ratio in the worst case. Our lower bound applies to a class of linear scaling based solutions which contains all but one known algorithm (which is also based on linear scaling but requires a different argument). Despite faster Swiper and extended Swiper being great "heuristic" schemes, we would still like to find an algorithm with better theoretical guarantees. We make some progress with this in subsequent sections.

Section 7 proves several facts about our weight reduction problem useful in the rest of the paper. Of particular importance is a new observation that assuming the weights $w_1, ..., w_n$ in the input are sorted, there exists an optimal solution $t_1, ..., t_n$ that is also sorted. In section 8 we show that this fact yields the first polynomial time algorithm with approximation factor $n/\left(c \log^2 n\right)$, for any $c > 0$, and the first exact (approximation factor 1) solution with time complexity $O\left(n + R^{3/2} e^{C\sqrt{R}}\right)$, where R is the cost of an optimal solution and $C \approx 2.57$. Given that R is often much smaller than n, our exact algorithm turns out to be practical for some real-world weight distributions, as shown by numerical evaluations. We believe it makes sense to use it in production when the optimal solution is expected to be small, in combination with a fallback scheme if the exact algorithm takes too long. It can also be very useful for research purposes, for example, to test some hypothesis about the problem.

Section 9 contains numerical evaluations comparing old and new algorithms in terms of the total weight in the output and the running time. It confirms our claims about the improved running time of faster Swiper and the improved output quality of extended Swiper. It also proves our sub-exponential time exact algorithm to be somewhat practical.

Finally, Appendix A describes a polynomial time algorithm based on the standard approach of representing an optimization problem as an integer linear program, relaxing it to allow fractional solutions, running an LP solver and rounding the solution to an integer vector in a problem specific manner. When denoting the cost of an optimal solution by $\mathsf{OPT}_{\leq\alpha,<\beta}$, our algorithm guarantees a solution of cost at most $\mathsf{OPT}_{\leq\alpha,<(1-\delta)\beta}$ for any arbitrarily small constant $\delta > 0$. In the full version of our paper [42, Appendix B] we, however, prove that this can be $\Omega(n)$ times larger than the optimal solution. Nevertheless, we believe the algorithm presents theoretical interest useful for future research. Moreover, such a guarantee can be meaningful when the protocol relying on committee selection can work with a continuous range of the final adversarial fractions β. Take for example Approximate Lower Bound Arguments [16] that allow one to create short threshold signatures. Its decentralized version uses committee selection and presents a configurable tradeoff between the size of the committee and the size of the final proof (e.g., a threshold signature) where the proof size would be some monotone function of β in our notation. Their committee selection algorithm is agnostic to the weight distribution, but suppose we replace it with theoretically smallest committees for any given β and plot the resulting tradeoff with the committee size on the horizontal axis and the proof size on the vertical axis. We would like to achieve this tradeoff, but since the optimal committee selection is likely infeasible, we are forced to use an approximation and get a worse curve. Using algorithms approximating the committee size for a fixed β, such as the one in subsection 8.2, would shift the tradeoff graph to the right while algorithms approximating β for a fixed committee size, such as the one in Appendix A, would shift the graph up. In the absence of a good approximation on the committee size, the second transformation could even be superior.

2 General overview of committee selection

The idea of selecting small committees can be seen as far back as [9]. It has since been used in constructing many efficient protocols for consensus [35, 36, 34, 10, 7]. For example, the proof-of-stake blockchain protocol Algorand [30, 17] modifies a classical setting $O(n^2)$ byzantine fault tolerant consensus protocol to run on committees created by selecting each unit of stake with some probability, which makes communication complexity per party independent of the number of parties. Additionally, each step of the BFT sub-protocol uses a newly sampled anonymous committee that reveals itself only when the members of the committee speak, thus achieving security against an adaptive adversary. Such an adversary can corrupt participants at any time of the protocol based on the information he possesses thus far, up to some threshold of total corrupted stake.

Committee selection can also be seen in communication efficient threshold signatures [15, 16, 24], leader election [37, 33], secure multiparty computation [8, 5, 19, 6, 18], and distributed key-generation [12].

Schemes utilizing committees typically possess a gap between the absolutely necessary requirement on the fraction of adversarial power and the smaller fraction assumed by the protocol. For instance, solving consensus in an asynchronous environment requires adversarial weight to be strictly less than 1/3 of the total weight while Algorand assumes that at most 1/5 of the total stake is controlled by the adversary. Motivated by proof-of-stake blockchains, we are interested in constructing small committees in the *weighted* setting. Given original weights $w_1, w_2, ..., w_n \in \mathbb{N}$ for n parties such that the adversary controls parties of total weight at most $\alpha \sum_{i=1}^{n} w_i$, we need to compute new weights $t_1, t_2, ..., t_n \in \mathbb{Z}_{\geq 0}$ such that the adversary can control strictly less than $\beta \sum_{i=1}^{n} t_i$ of new weight.

There are at least two dimensions of interesting variants of this problem. One is the definition of the objective function. A party in Algorand sends out one vote per step of the protocol even if it has multiple units of stake on the committee. Thus, for Algorand, we are interested only in the number of distinct parties on the committee or, equivalently, the number of non-zero t_i. On the other hand, there are schemes that cannot efficiently aggregate weights, say Shamir secret sharing. For such protocols, it may be necessary to use "virtualization" – essentially, giving w virtual identities to a party with weight $w \in \mathbb{Z}_{\geq 0}$ and running this many copies of the protocol. This can be prohibitively expensive when weights are large (e.g., in proof-of-stake blockchains weights can be on the order of 2^{64}), and thus, for such protocols we are interested in constructing committees with $\sum_{i=1}^{n} t_i$ as small as possible – the goal we pursue in this paper. We note that there have been many works trying to make cryptographic protocols scale better with the weights [3, 4, 27, 20, 15, 16, 24]. Still, protocols such as those in [22] only improve the dependence on weights for the verifiers while the provers' running time is still linear in the weights, and protocols such as those in [26] only improve complexities to $O(W)$ down from $O(W \cdot \lambda)$, for a total weight W and security parameter λ.

Another configuration knob for the problem of committee selection is, what kind of security do we require? Traditionally, especially in a setting where the adversary is static, meaning he chooses parties to corrupt before the start of the protocol, a random committee is chosen such that the (unknown) corrupted set of parties owns only a small fraction of the elected weight. More formally, the security requirement of this "randomized" variant is that

$$\begin{aligned} &\forall (w_1, ..., w_n),\ \forall A \subseteq [n] \text{ with } \sum_{i \in A} w_i \leq \alpha \sum_{i=1}^{n} w_i, \\ &\Pr\left[\sum_{i \in A} t_i < \beta \sum_{i=1}^{n} t_i : (t_1, ..., t_n) \leftarrow \text{ComputeCommittee}(w_1, ..., w_n)\right] \text{ is large} \end{aligned} \tag{1}$$

(we assume that α and β are constants). Instead, this work considers a stronger security notion where the computed committee must be secure *for any* subset of the parties that can be corrupted. A more formal statement of this "deterministic" version of the problem is as follows.

$$\begin{aligned} &\forall (w_1, ..., w_n), \quad \text{if } (t_1, ..., t_n) = \text{ComputeCommittee}(w_1, ..., w_n) \text{ then} \\ &\forall A \subseteq [n] \text{ with } \sum_{i \in A} w_i \leq \alpha \sum_{i=1}^{n} w_i,\ \sum_{i \in A} t_i < \beta \sum_{i=1}^{n} t_i \end{aligned} \tag{2}$$

(where ComputeCommittee is now deterministic). When the weaker security notion is sufficient, it is great news since selecting each unit of weight with some equal probability is a simple and efficient scheme yielding an expected constant size committee; this is generally impossible for a deterministic algorithm (since n parties with equal weights must elect a committee of size $\Omega(n)$, lest the entire committee be corrupted). On the other hand, deterministic committee selection makes it easy to maintain security against an adversary who is allowed to corrupt parties adaptively throughout the protocol execution since any corrupted set will have less than β fraction of new weight in the committee. While simple quorum systems can get away with using random committees with VRF as a source of secret randomness, other problems are less trivial and require intricate protocols and arguments of security [17, 29, 14, 11, 1]. The security against adaptive corruptions has been recognized important by the community, and in fact, is the reason why [22, 21] has been deployed in the Aptos blockchain in combination with a deterministic committee selection scheme (from

private communication with the authors). Moreover, there is evidence that a deterministic scheme can produce smaller committees than a randomized one. Take, for example, the standard randomized procedure of selecting each unit of weight with some equal probability. As the requirement on how large the probability in Equation 1 must be increases, the expected number of selected units of weight approaches infinity and the expected number of elected parties approaches n. Thus, for a sufficiently large mandated success probability, the chosen committee would be worse than that of a deterministic algorithm. Another example is the randomized schemes presented in [28] called Fait Accompli that include larger parties in the committee deterministically to either reduce the number of parties in the committee or their total new weight. From their formulas and plots, one can see that as the required probability of failure decreases, the committee must grow. Indeed, it seems inherent that when we require $\sum_{i\in A} t_i$ in Equation 1 to be increasingly concentrated, its expectation must approach infinity; it would be a good exercise to prove that as the probability of failure goes to 0, whenever at least one t_i is non-deterministic, $E\left[\sum_{i=1}^{n} t_i\right]$ must approach infinity and $E\left[|\{i : t_i > 0\}|\right]$ must approach or exceed the number of parties in an optimal deterministic committee.

In our work, we focus on the "deterministic" flavor of committee selection with the objective to minimize the total weight $\sum_{i=1}^{n} t_i$, and believe this problem is important for the reasons outlined above. Our paper can be viewed as a continuation of the Fait Accompli work [28] by Gaži, Kiayias and Russell and the Swiper work [38] by Tonkikh and Freitas that frame committee selection as an optimization problem with parties' weights in the input. Fait Accompli shows randomized schemes that optimize either $\sum_{i=1}^{n} t_i$ or $|\{i : t_i > 0\}|$, while Swiper describes algorithms for the same problem we consider, and in fact, we directly extend some of their results. Despite both works numerically showing impressive results on real-world, far from uniform, weight distributions, they say nothing about the quality of their algorithms' output compared to the best possible solution. The golden standard in approximation algorithms is to provide one that yields solutions of the cost within a certain multiplicative factor of the cost of an optimal solution [40], or otherwise bounded by some other "interesting" function of the input (as in [32] for ECSS). We make some progress towards this.

Finally, we note that there is no reason to require algorithms satisfying the stronger security notion in Equation 2 to be deterministic. The use of randomness in optimization often helps [40], but all algorithms in this paper are in fact deterministic.

3 Problem definition

Let $n \in \mathbb{N}$ be the number of parties, $w_1, ..., w_n \in \mathbb{N}$ be their (original) weights, and let α and β be constants independent of n with $0 < \alpha < \beta < 1$. Borrowing notation from [38], for the new weights $t_1, ..., t_n \in \mathbb{Z}_{\geq 0}$, we say party i gets t_i tickets and call $(t_1, ..., t_n)$ a ticket assignment. We call $(t_1, ..., t_n)$ valid if and only if $\sum_{i\in I} t_i < \beta \sum_{i=1}^{n} t_i$ whenever $I \subseteq [n]$ and $\sum_{i\in I} w_i \leq \alpha \sum_{i=1}^{n} w_i$. Our goal is to find a valid ticket assignment that minimizes the total number of tickets $\sum_{i=1}^{n} t_i$. We call this the $(\leq \alpha, < \beta)$-weight reduction problem and denote $\mathsf{OPT}_{\leq\alpha,<\beta}((w_1, ..., w_n))$ to be the smallest total number of tickets $\sum_{i=1}^{n} t_i$ in a valid solution $(t_1, ..., t_n)$. We can also formulate the problem using the following integer linear program.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} t_i \\
\text{subject to} \quad & \sum_{i\in I} t_i < \beta \sum_{i=1}^{n} t_i \quad \forall I \subseteq [n] \text{ s.t. } \sum_{i\in I} w_i \leq \alpha \sum_{i=1}^{n} w_i \\
& t_i \in \mathbb{Z}_{\geq 0} \qquad\qquad \forall i \in [n]
\end{aligned}$$

We note that the formulation in [38] uses a strict inequality where α is involved. For simplicity, especially when dealing with linear programming, we use a non-strict inequality. However, we still keep a strict inequality where β is involved since we are typically interested in solutions where the adversary controls strictly less than $1/2$ or $1/3$ of the total number of tickets and formulating the problem using a strict inequality simplifies asymptotic analysis where n approaches infinity. We call the variant in [38] the $(<\alpha, <\beta)$-weight reduction problem with optimum $\mathsf{OPT}_{<\alpha,<\beta}((w_1, ..., w_n))$, and also similarly define $(\leq\alpha, \leq\beta)$- and $(<\alpha, \leq\beta)$-weight reduction problems that have an additional requirement $\sum_{i=1}^{n} t_i \geq 1$.

We also note that we do not allow zero weights w_i in the input, but it does not change the problem since it is always optimal to set $t_i := 0$ for such i.

4 Detailed overview of Swiper

We now give a more detailed overview of the Swiper algorithm from [38, 39] since we will be modifying it later in section 5.

It works by linearly scaling down the weights and rounding to an integer according to a rule. Specifically, given weights $w_1, ..., w_n$ as input, Swiper outputs $t_1, ..., t_n$ such that for all i, either $t_i = \lfloor sw_i + c \rfloor$ or $t_i = \lfloor sw_i + c \rfloor - 1$, where c is a constant to be defined later. Swiper first runs a binary search on s to find a local minimal value for which the ticket assignment with all $t_i = \lfloor sw_i + c \rfloor$ is valid. Then the algorithm considers those i for which $(sw_i + c)$ is an integer, or equivalently those i for which t_i decreases when s is decreased by any value; call this borderline set of indices B. Finally, Swiper runs a binary search on $k \in \mathbb{N}$ to find a local minimal value in $\left[|B|\right]$ for which the ticket assignment remains valid when all but k arbitrary indices from B lose one ticket.

From a slightly different perspective, Swiper defines a set of potential solutions $\vec{t_1}, \vec{t_2}, ...$ with $\sum_{i=1}^{n} \left(\vec{t_j}\right)_i = j$ and finds a local minimal j for which $\vec{t_j}$ is a correct solution but $\vec{t_{j-1}}$ is not.

The authors observe that starting some index $M \in O(n)$, all $\vec{t_M}, \vec{t_{M+1}}, ...$ are valid. In particular, when $c = \alpha$, we have $M = \left\lfloor \frac{\alpha(1-\alpha)}{\beta-\alpha} n + 1 \right\rfloor$ (for the $(\leq\alpha, <\beta)$ weight reduction variant of the problem). By making M the right threshold of the search, Swiper always outputs a solution with at most this many tickets, though a much better solution is usually found for real-world weight distributions.

Each iteration of a binary search needs to test whether some ticket assignment $(t_1, ..., t_n)$ is valid. This is done precisely by solving the knapsack problem where weights are w_i and profits are t_i. Since Swiper only tests ticket assignments of size at most $M = O(n)$, the knapsack problem can be solved in time $O\left(n \cdot \sum_{i=1}^{n} t_i\right) = O\left(n^2\right)$ using dynamic programming that for each $0 \leq t \leq \sum_{i=1}^{n} t_i$ computes the smallest possible weight that achieves t tickets. The total time complexity of Swiper is $O(n^2 \log n)$.

5 Improving Swiper

In this section we present two variants of Swiper, one that improves the running time of the algorithm and the other that improves the output at the cost of a reasonable increase to the running time.

5.1 Faster Swiper

We first show a new algorithm named faster Swiper that improves the asymptotic running time of the original Swiper algorithm without semantically changing the output. Like the original Swiper, faster Swiper finds a ticket assignment $\vec{t}$ such that

- $\vec{t}$ is valid, equals a linear scaling of the weights according to Swiper rules (see section 4), and $\sum_i \left(\vec{t}\right)_i$ satisfies the $O(n)$ bound of the original Swiper;
- there exists a ticket assignment $\vec{t'}$ with one fewer ticket such that it equals a linear scaling of the weights according to Swiper rules but is not valid.

The stated time complexity of Swiper in [38] is $O(n^2 \log n)$ where n is the number of weights in the input. Our algorithm faster Swiper has time complexity $O(n + R^2)$ where R is the cost of the output (i.e., if the output is $\vec{t}$, $R = \sum_i \left(\vec{t}\right)_i$). Note that R is always $O(n)$ but at the same time, R can be much smaller than n in practice, and thus, looking at the asymptotic complexity in terms of two variables n and R can be useful.

It will be convenient to have access to the input weights vector sorted in the descending order. Let us assume such sorted input $(w_1, w_2, ..., w_n)$. We first explain how to achieve running time $O(R^2 \log R)$ and then improve it further to $O(R^2)$. In subsection 5.3 we will explain how to remove the assumption.

5.1.1 $O(R^2 \log R)$ time complexity

For all $i \geq 0$, let $\vec{t_i}$ be the vector with i tickets in total constructed according to Swiper rules such that for any i, $\vec{t_i}$ and $\overrightarrow{t_{i+1}}$ have the same values for all indices except for one; when multiple indices could be incremented in the transition from $\vec{t_i}$ to $\overrightarrow{t_{i+1}}$, choose the lowest index. The (uniquely defined) sequence $\left\{\vec{t_i}\right\}$ has a property that if $\left(\vec{t_i}\right)_j > 0$ then $\left(\vec{t_i}\right)_k > 0$ for all $k \leq j$. This in turn implies that if the cost of $\vec{t_i}$ is c, only the first c indices of $\vec{t_i}$ can be non-zero. Thus, if an algorithm only ever considers ticket assignments of cost at most $O(R)$, it only needs to query the first $O(R)$ elements of the (sorted) weight vector.

We make use of this fact. To get time complexity $O(R^2 \log R)$, we run binary search on i to find a value for which $\vec{t_i}$ is a valid ticket assignment but $\overrightarrow{t_{i-1}}$ is not, roughly as follows.

```
l ← 0; r ← 1;
while t_r⃗ is not valid do
    l ← r; r ← 2 · r;
while r − l > 1 do
    m ← (l + r)/2;
    if t_m⃗ is valid then
        r ← m;
    else
        l ← m;
return t_r⃗;
```

This requires $O(\log R)$ iterations of binary search, unlike in the original Swiper that does binary search on the scaling factor s potentially requiring $\Omega(\log n)$ iteration. Each iteration needs to solve a knapsack problem using dynamic programming to determine whether some $\vec{t_i}$ is valid. The time required to do this is at most the total number of tickets times the number of non-zero entries in $\vec{t_i}$, since the zero entries cannot help the adversary. Both are at most i. Hence, the first loop will run in time $O(R^2)$ and the second loop in time $O(R^2 \log R)$.

We are left to describe how to efficiently calculate $\vec{t_0}, \vec{t_1}, ..., \overrightarrow{t_{2R}}$. We use a mutable data structure called gen with a single method gen.next() such that if the i-th invocation of gen.next() returns index, then $\left(\vec{t_i}\right)_{\text{index}} = \left(\overrightarrow{t_{i-1}}\right)_{\text{index}} + 1$ (and all other indices stay the same). Such a data structure can be constructed using a binary heap as a priority queue defining the order of the produced indices by gen.next(). The heap initially contains only the first index – the index of the largest weight. When gen.next() outputs some index, this index is reinserted in the heap; if index has never been produced before, index + 1 is additionally

inserted. Therefore, during the k-th call to gen.next(), the heap contains at most $2k$ elements and thus, the call takes time $O(\log k)$. Our Swiper variant needs to query gen at most $2R$ times, and hence, will add $O(R \log R)$ to the time complexity.

5.1.2 $O(R^2)$ time complexity

To improve the running time of faster Swiper further, we notice that iterations of the binary search can reuse some of the dynamic programming (DP) computations. The structure of the binary search remains the same as in the $O(R^2 \log R)$ version, and we only need to optimize its main phase (the second while loop) since the initial phase already runs in time $O(R^2)$.

We need a data structure called DP containing (partial) DP computations for the knapsack problem. Specifically, it will have methods DP.apply(w, t) and DP.get() such that after running DP.apply on a sequence $(w_1, t_1), ..., (w_k, t_k)$, DP.get() reports the maximum number of tickets that the adversary can get $\sum_{j \in S} t_j$ provided $\sum_{j \in S} w_j \le \alpha W$ ($W = \sum_i w_i$ being the total number of tickets in the input). Since the binary search only tests ticket assignments $\vec{t_i}$ of cost at most $2R$, DP.apply will always run in time $O(R)$. Additionally, a complete DP computation requires calling DP.apply the number of times at most the number of non-zero elements of $\vec{t_i}$ which is at most $2R$.

The idea is to reuse some DP computation to have each iteration of the binary search (second loop) run in time $O(R(r-l))$. Since $(r-l)$ halves each iteration, the entire algorithm will be $O(R^2)$.

The loop will have the following invariants holding at the beginning of the loop.

- tickets_l: array containing the non-zero prefix of $\vec{t_l}$;
- deltas: array containing the indices from gen.next() to generate $\overrightarrow{t_{l+1}}, ..., \vec{t_r}$;
- dp_head: DP data structure with applied $\left(w_i, \left(\vec{t_l}\right)_i\right)$ for all $i \notin$ deltas.

Essentially, dp_head will have indices applied whose values are not going to change anymore. Then testing the validity of $\vec{t_m}$ requires computing it in time $O(R)$ and applying the missing indices, contained in deltas, to a copy of dp_head in time $O(R(r-l))$. After that, the loop iteration needs to update the variables to maintain the invariants: tickets_l in time $O(R)$, deltas in constant time and dp_head in time $O(R(r-l))$ by applying a subset of indices from deltas.

5.2 Extended Swiper

We will now present a variant of Swiper that we call extended Swiper which replaces the binary search on the number of tickets in the solution with a linear search.

$i \leftarrow 1$;
while $\vec{t_i}$ is not valid **do**
  $i \leftarrow i + 1$;
return $\vec{t_i}$;

The linear search sometimes significantly improves the output in practice since the binary search gets stuck in a "local minimum". Although the idea is trivial, our contribution is an algorithm that runs the linear search in time $O(R^{2.5})$ instead of the easy $O(R^3)$, assuming the input weights are sorted.

The idea is to reuse DP computation as in subsection 5.1. We check the validity of $\vec{t_i}$ in batches. The batch number $j \ge 1$ will be $(\overrightarrow{t_{i_j+1}}, ..., \overrightarrow{t_{i_{j+1}}})$ where $i_1 = 0$ and $i_{j+1} = i_j + \Theta\left(\sqrt{i_j}\right)$. The batch j will be processed in time $O\left(i_j^2 + (i_{j+1} - i_j)^2 \cdot i_{j+1}\right) = O\left(i_j^2\right)$.

Before describing the batch algorithm, we briefly argue why the time complexity of the whole algorithm is $O(R^{2.5})$. Take any $d \in \mathbb{N} \cup \{0\}$ and let j be the largest integer such that $i_j + 1 \leq 2^d$ and k the smallest integer such that $i_k \geq 2^{d+1} - 1$. The batches $j, ..., (k-1)$, $\Theta(2^{d/2})$ in total, will each be processed in time $O(2^{2d})$ and thus together in time $O(2^{2.5d})$. Hence, the whole algorithm works in time $O\Big(\sum_{d=0}^{\lceil \log R \rceil} 2^{2.5d}\Big) = O\big(2^{2.5 \log R}\big) = O\big(R^{2.5}\big)$.

We now describe the batch algorithm. For batch j, collect indices from gen.next() that generate $\overrightarrow{t_{i_j+1}}, ..., \overrightarrow{t_{i_{j+1}}}$ into an array deltas. Create a DP data structure dp_head and call dp_head.apply on $\Big(w_k, \big(\overrightarrow{t_{i_j}}\big)_k\Big)$ for all $k \notin$ deltas, this takes time $O(i_j^2)$. Now for each index in deltas, apply it to get the next $\overrightarrow{t_i}$ and test its validity by applying $\Big(w_k, \big(\overrightarrow{t_i}\big)_k\Big)$ for distinct $k \in$ deltas to a copy of dp_head. This takes time $O\big((i_{j+1} - i_j) \cdot i_{j+1} \cdot (i_{j+1} - i_j)\big)$. Hence, the batch algorithm has running time $O\big(i_j^2 + (i_{j+1} - i_j)^2 \cdot i_{j+1}\big)$.

5.3 Arbitrarily ordered weights

We previously assumed that the weights vector in the input is sorted. We will now explain how to remove this assumption for an additive cost of $O(n + R \log R)$.

We need a mutable data structure that we call sorted_weights that has a single method sorted_weights.next() returning the next largest weight from the input. We require that initializing sorted_weights takes time $O(n)$ and that the first m calls to sorted_weights.next() take cumulative time $O(m \log m)$. This is sufficient since our algorithms would query sorted_weights.next() only $O(R)$ times.

Such a data structure can be constructed using a binary max-heap initialized in time $O(n)$ to contain all the weights and an auxiliary max-heap initially containing the reference to the main heap's root. sorted_weights.next() returns the root of the auxiliary heap and inserts in the auxiliary heap the references to the two children of the returned element in the main heap. [25]

5.4 A note on reducing time complexity further

Swiper [39] explores using a quasilinear time approximate knapsack solver in lieu of the quadratic time exact algorithm. Numerical evaluations by the authors show that this reduces the quality of the output only by a little. While we do not explore this modification, it could be interesting to employ it to potentially reduce the running time of faster Swiper and extended Swiper to $O(n) + \tilde{O}(R)$ and $O(n) + \tilde{O}(R^{1.5})$ respectively.

6 Analysis of linear scaling based algorithms

We are interested in analyzing the approximation factor of known algorithms solving the $(\leq \alpha, < \beta)$-weight reduction problem, meaning how many times larger can the computed solution be compared to an optimal solution. We notice that all previously known algorithms use linear scaling with rounding to compute ticket assignments $(t_1, ..., t_n)$ and prove that all have worst case approximation factor $\Omega(n)$. We start with the following theorem that applies to a class of linear scaling based solutions.

▶ **Theorem 1.** *Let $\alpha, \beta \in \mathbb{R}$ and $r, s \in \mathbb{N}$ such that $0 < \alpha < \beta < 1$ and $\frac{r}{s+1} < \beta \leq r/s \leq 1$. Also let $0 \leq c < 1$, $m \in \mathbb{Z}_{\geq 0}$, and suppose an algorithm solving the $(\leq \alpha, < \beta)$-weight reduction problem on input $(w_1, ..., w_n)$ always outputs $(t_1, ..., t_n)$ with the property that there*

exists $\delta > 0$ such that for all i,

$$\lfloor \delta w_i + c \rfloor - m \leq t_i \leq \lfloor \delta w_i + c \rfloor.$$

Then for infinitely many n there exists a weight vector $\vec{w} = (w_1, ..., w_n)$ such that

$$OPT_{\leq \alpha, < \beta}(\vec{w}) = s + 1$$

but the algorithm outputs a solution with the number of tickets

$$> \big(1 - c\big) \frac{\alpha n - r - 1}{r/s - \alpha} - ms.$$

The worst case approximation factor of the algorithm is thus

$$> \frac{(1-c)\alpha}{(s+1)(r/s - \alpha)} n - O(1).$$

Proof. Define $\vec{w} = (w_1, ..., w_n)$ where $w_1 = w_2 = ... = w_{n-s} = 1$ and $w_{n-s+1} = ... = w_n = w$ where

$$w = \left\lfloor \frac{\alpha n - \alpha s}{r - \alpha s} \right\rfloor = \frac{\alpha n - \alpha s}{r - \alpha s} - \varepsilon \tag{3}$$

for some $0 \leq \varepsilon < 1$. The total weight is $\sum_{i=1}^{n} w_i = n - s + sw$ and the adversary is allowed to corrupt the following weight.

$$W_{\mathsf{A}} = \lfloor \alpha W \rfloor = \lfloor \alpha n - \alpha s + \alpha s w \rfloor = \left\lfloor \alpha n - \alpha s + \frac{\alpha^2 sn - \alpha^2 s^2}{r - \alpha s} - \alpha s \varepsilon \right\rfloor =$$
$$\left\lfloor \frac{\alpha r n - \alpha r s}{r - \alpha s} - \alpha s \varepsilon \right\rfloor = \frac{\alpha r n - \alpha r s}{r - \alpha s} - \alpha s \varepsilon - \varepsilon'$$

for some $0 \leq \varepsilon' < 1$.

Then $W_{\mathsf{A}} - rw = (r - \alpha s)\varepsilon - \varepsilon'$. We would like to identify an infinite sequence of n such that $W_{\mathsf{A}} = rw$. Since $r - \alpha s > 0$, we have $W_{\mathsf{A}} - rw > -1$. If $r - \alpha s \leq 1$, then also $W_{\mathsf{A}} - rw < 1$ and we are done. Otherwise, only consider $n = s + \lceil k \cdot \frac{r - \alpha s}{\alpha} \rceil$ for $k \in \mathbb{N}$. Then $n = s + k \cdot \frac{r - \alpha s}{\alpha} + \varepsilon''$ for some $0 \leq \varepsilon'' < 1$ and $\alpha(n - s) = k(r - \alpha s) + \alpha \varepsilon''$. From Equation 3,

$$\varepsilon = \frac{\alpha n - \alpha s}{r - \alpha s} - w = \frac{k(r - \alpha s) + \alpha \varepsilon''}{r - \alpha s} - w = k - w + \frac{\alpha \varepsilon''}{r - \alpha s}.$$

Since $k - w \in \mathbb{Z}$, $0 \leq \varepsilon < 1$ and $0 \leq \frac{\alpha \varepsilon''}{r - \alpha} < 1$, we have $\varepsilon = \frac{\alpha \varepsilon''}{r - \alpha s} < \frac{1}{r - \alpha s}$. Hence, $W_{\mathsf{A}} - rw < 1$ and $W_{\mathsf{A}} = rw$.

Consider a ticket assignment $\vec{t} = (t_1, ..., t_n)$ where $t_1 = ... = t_{n-s-1} = 0$ and $t_{n-s} = ... = t_n = 1$. Since the adversary can only corrupt r parties that have a ticket, the total number of tickets is $s + 1$ and $r < \beta(s + 1)$ by an assumption in the theorem statement, this is a valid solution of cost $s + 1$. Clearly, no better solution is possible since the adversary can always corrupt r tickets (assuming they exist) to get adversarial ratio $\geq r/s$. Hence, $\mathsf{OPT}_{\leq \alpha, < \beta}(\vec{w}) = s + 1$.

We will now prove a lower bound on the cost of algorithm's output. Take any valid ticket assignment $\vec{t} = (t_1, ..., t_n)$ and notice that at least one of t_1, ..., t_{n-s} must be non-zero; otherwise the adversary can corrupt r parties with the largest number of tickets to get

adversarial ratio $\geq r/s$. Since the algorithm chooses $\delta > 0$ and for all i outputs $t_i \leq \lfloor \delta w_i + c \rfloor$, we must have $\delta \geq 1 - c$. Hence, the algorithm outputs a ticket assignment of cost at least

$$s\big(\lfloor (1-c)w + c \rfloor - m\big) > s\big((1-c)w + c - 1 - m\big) >$$
$$s\left(\big(1-c\big)\frac{\alpha n - \alpha s - 1}{r - \alpha s} - (1-c) - m\right) = s\left(\big(1-c\big)\frac{\alpha n - r - 1}{r - \alpha s} - m\right) =$$
$$\big(1-c\big)\frac{\alpha n - r - 1}{r/s - \alpha} - ms. \qquad \blacktriangleleft$$

When β is rational, simply set r and s such that $\beta = r/s$ to apply the lower bound. Otherwise, we need to find r, s such that $\frac{r}{s+1} < \beta \leq \frac{r}{s} \leq 1$ or, equivalently, $1 \leq \frac{s}{r} \leq \frac{1}{\beta} < \frac{s+1}{r}$. Setting $s = \lfloor 1/\beta \rfloor$ and $r = 1$ satisfies the inequality.

We now discuss particular weight reduction algorithms. By a simple modification of an argument in [39], we know that Swiper that solves the $(\leq \alpha, < \beta)$-weight reduction problem always outputs a solution with at most $\frac{\alpha(1-\alpha)}{\beta-\alpha}n + 1$ tickets. Thus, this is also an upper bound on its approximation factor. Since Swiper outputs solutions $(t_1, ..., t_n)$ such that for each i, $t_i = \lfloor \delta w_i + \alpha \rfloor$ or $t_i = \lfloor \delta w_i + \alpha \rfloor - 1$, by Theorem 1, the worst case approximation factor of Swiper is $\frac{\alpha(1-\alpha)}{(s+1)(r/s-\alpha)}n - O(1)$ (for appropriate r and s). Notice, that for some values of β, this is very close to the upper bound; for instance, when $\beta = 1/2$, it is a factor 3 away from the upper bound. The same analysis applies to faster Swiper and extended Swiper.

We now discuss the gcdWR algorithm from [23] (also see an overview in the full version of our paper [42, Appendix A]). It is quite similar to Swiper in that it outputs a ticket assignment $(t_1, ..., t_n)$ of the form $t_i = \lfloor \delta w_i + c \rfloor$; here, $\delta = 1/\gcd$ and $c = 1/2$. Theorem 1 gives approximation factor lower bound of $\frac{\alpha}{2(s+1)(r/s-\alpha)}n - O(1)$. We conclude that the worst case approximation factor of gcdWR is $\Theta(n)$.

Finally, we discuss the algorithms in [21, v1] and [4]. Even though they solve other variants of the weight reduction problem, we can still see what approximation factor they would give if solutions of the same form were to be used for the $(\leq \alpha, < \beta)$-weight reduction problem. [21, v1] outputs solutions of the form $t_i = \lfloor \delta w_i + 1/2 \rfloor$. By Theorem 1, its worst case approximation factor exceeds $\frac{\alpha}{2(s+1)(r/s-\alpha)}n - O(1)$ (for appropriate r and s).

[4] outputs solutions of the form $t_i = \lceil \delta w_i \rceil$. Since each w_i is positive, each t_i also is and thus the algorithm always outputs a solution of cost at least n. We are left to find an example where the optimal solution is small. If there exists a weight w_i large enough that the adversary cannot corrupt it, then the optimal solution has one ticket in total. Hence, the worst case approximation factor is at least n.

7 Useful facts

In this section we present several facts that will be useful in the rest of the paper. We start with a (new) characterization of optimal solutions.

▶ **Theorem 2.** *Assume $w_1 \leq w_2 \leq ... \leq w_n$. There exists an optimal solution $(t_1, ..., t_n)$ such that $t_1 \leq t_2 \leq ... \leq t_n$.*

Proof. Let $(t_1, ..., t_n)$ be an optimal solution such that $t_i > t_j$ for some $i < j$. We will show that $(t'_1, ..., t'_n) = (t_1, ..., t_{i-1}, t_j, t_{i+1}, ..., t_{j-1}, t_i, t_{j+1}, ...t_n)$ is also an optimal solution (i.e., t_i and t_j are swapped). This has the same cost, we just need to show that it is a valid ticket assignment.

Let $I \subseteq [n]$ such that $\sum_{k \in I} w_k \leq \alpha \sum_{k=1}^{n} w_k$. Consider the following cases.

- $i \notin I$, $j \notin I$: $\sum_{k \in I} t'_k = \sum_{k \in I} t_k < \beta \sum_{k=1}^{n} t_k = \beta \sum_{k=1}^{n} t'_k$;

- $i \in I$, $j \in I$: $\sum_{k\in I} t'_k = \sum_{k\in I} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$;
- $i \in I$, $j \notin I$: $\sum_{k\in I} t'_k < \sum_{k\in I} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$;
- $i \notin I$, $j \in I$: let $I' = I \setminus \{j\} \cup \{i\}$, $\sum_{k\in I} t'_k = \sum_{k\in I'} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$, where the inequality holds since $\sum_{k\in I'} w_k < \sum_{k\in I} w_k \le \alpha \sum_{k=1}^n w_k$.

Hence, $(t'_1, ..., t'_n)$ is a valid assignment.

Therefore, we can sort all t_i using, for example, insertion sort to get an optimal sorted assignment $(t_1, ..., t_n)$. ◀

We also prove a slightly stronger statement.

▶ **Theorem 3.** *Assume $w_1 \le w_2 \le ... \le w_n$. There exists an optimal solution $(t_1, ..., t_n)$ such that $t_1 \le t_2 \le ... \le t_n$ and, moreover, for any i, j with $w_i = w_j$, $|t_i - t_j| \le 1$.*

We present the proof on page 24 of the Appendix.

We will also find useful the following, first mentioned in [39, v2 appendix C]. Similarly to the definition of $\mathsf{OPT}_{\le\alpha,<\beta}(\vec{w})$ from section 3, define $\mathsf{OPT}^{T_\mathsf{A}}_{\le\alpha,<\beta}(\vec{w})$ to be the smallest maximum number of tickets that the adversary can obtain in a valid solution:

$$\mathsf{OPT}^{T_\mathsf{A}}_{\le\alpha,<\beta}(\vec{w}) = \min_{\substack{(t_1,...,t_n)\in\mathbb{Z}^n_{\ge 0}\\ \text{is valid}}} \max_{\substack{I\subseteq[n]\wedge\\ \sum_{i\in I} w_i \le \alpha \sum_{i=1}^n w_i}} \sum_{i\in I} t_i.$$

▶ **Theorem 4.** *For all $0 < \alpha < \beta < 1$ and $\vec{w} \in \mathbb{N}^n$, we have*

$$OPT_{\le\alpha,<\beta}(\vec{w}) = \left\lfloor \frac{OPT^{T_A}_{\le\alpha,<\beta}(\vec{w})}{\beta} \right\rfloor + 1; \quad OPT^{T_A}_{\le\alpha,<\beta}(\vec{w}) = \left\lceil \beta \cdot OPT_{\le\alpha,<\beta}(\vec{w}) \right\rceil - 1.$$

Proof. Denote $T = \mathsf{OPT}_{\le\alpha,<\beta}(\vec{w})$ and $T_\mathsf{A} = \mathsf{OPT}^{T_\mathsf{A}}_{\le\alpha,<\beta}(\vec{w})$.

We first prove that $T = \lfloor T_\mathsf{A}/\beta \rfloor + 1$. Let $(t_1, ..., t_n)$ be a valid ticket assignment where the maximum total number of tickets the adversary can obtain is T_A. Decrease all t_i as much as possible as long as $T_\mathsf{A} < \beta \sum_{i=1}^n t_i$; call the resulting ticket assignment $(t'_1, ..., t'_n)$, which must be valid. Then $T \le \sum_{i=1}^n t'_i = \lfloor T_\mathsf{A}/\beta \rfloor + 1$. Now take a valid ticket assignment $(t_1, ..., t_n)$ with $\sum_{i=1}^n t_i = T$. We have

$$T_\mathsf{A} \le \max_{\substack{I\subseteq[n]\wedge\\ \sum_{i\in I} w_i \le \alpha \sum_{i=1}^n w_i}} \sum_{i\in I} t_i < \beta T$$

and thus $T \ge \lfloor T_\mathsf{A}/\beta \rfloor + 1$. Hence, $T = \lfloor T_\mathsf{A}/\beta \rfloor + 1$.

We will now prove that $T_\mathsf{A} = \lceil \beta T \rceil - 1$. Since $T = \lfloor T_\mathsf{A}/\beta \rfloor + 1$, $T > T_\mathsf{A}/\beta$ which implies $T_\mathsf{A} < \beta T$ which implies $T_\mathsf{A} \le \lceil \beta T \rceil - 1$. Also since $T = \lfloor T_\mathsf{A}/\beta \rfloor + 1$, $T - 1 \le T_\mathsf{A}/\beta$ which implies $T_\mathsf{A} \ge \beta T - \beta \ge T_\mathsf{A} \ge \lceil \beta T \rceil - 1 - \beta$ which implies $T_\mathsf{A} \ge \lceil \beta T \rceil - 1$. ◀

Similarly, we have the following for the $(\le \alpha, \le \beta)$-weight reduction problem.

▶ **Theorem 5.** *For all $0 < \alpha < \beta < 1$ and $\vec{w} \in \mathbb{N}^n$, we have*

$$OPT_{\le\alpha,\le\beta}(\vec{w}) = \max\left\{1, \left\lceil \frac{OPT^{T_A}_{\le\alpha,\le\beta}(\vec{w})}{\beta} \right\rceil\right\}; \quad OPT^{T_A}_{\le\alpha,\le\beta}(\vec{w}) = \left\lfloor \beta \cdot OPT_{\le\alpha,\le\beta}(\vec{w}) \right\rfloor.$$

8 Optimized bruteforce

Given that all previously known weight reduction algorithms have a worst case approximation factor $\Omega(n)$, we seek algorithms with improved approximation. In this section we present the first sub-exponential exact algorithm that is practical for some real-world weight distributions, as well as the first polynomial time algorithm with approximation factor at most $n/(c\log^2 n)$ for any $c > 0$.

8.1 Sub-exponential time exact solution

In this section we show an algorithm that solves the $(\leq \alpha, < \beta)$-weight reduction problem exactly (i.e., that finds an optimal solution) in time $O\left(n + R^{3/2}e^{C\sqrt{R}}\right)$ where $C = \pi\sqrt{6}/3 \approx 2.57$ and $R \leq O(n)$ is the cost of the output (i.e., the optimal total number of tickets). Given that R is often relatively small and definitely smaller than n, this algorithm is sometimes practical for real-world distributions, as the numerical evaluations in section 9 show. Thus, when a small solution is expected, it might make sense to run the exact algorithm for a number of steps and switch to a fallback scheme if necessary. The algorithm could also be very useful for research purposes, for example, to test some hypothesis about the problem.

Assume the weights in the input are sorted. Using the characterization of optimal solutions in Theorem 2, the algorithm for each $i = 1, 2, ...,$ enumerates all sorted ticket assignments with total number i and for each, checks its validity using the knapsack algorithm. It outputs the first found valid ticket assignment.

The number of sorted ticket assignments with total number i is bounded by the number of ways $p(i)$ to represent i as a sum of positive integers, where p is called partition function[1]. It is known that $p(k) \sim \frac{1}{4\sqrt{3}k} \exp\left(\frac{\pi\sqrt{6k}}{3}\right)$. Also, the knapsack algorithm checks the validity of a ticket assignment in time $O(i^2)$ since the ticket assignment has at most i non-zero entries. Hence, the running time of this algorithm is $O\left(\sum_{i=1}^{R} i^2 p(i)\right)$ which is $O\left(R^{3/2}e^{C\sqrt{R}}\right)$ by the following lemma.

▶ **Lemma 6.** *For any $c > 0$, $\sum_{i=1}^{k} ie^{c\sqrt{i}} = O\left(k^{3/2}e^{c\sqrt{k}}\right)$.*

We present the proof on page 24 of the Appendix.

We note that it might be possible to reduce this time complexity by a factor of $R^{1/2}$ by reusing dynamic programming computations as in section 5. Lastly, see subsection 5.3 on how to handle unsorted inputs for an additive running time increase of $O(n + R\log R)$.

8.2 $O(n/\log^2 n)$-approximation in polynomial time

Using the idea in subsection 8.1 we can similarly construct an algorithm with approximation factor $\frac{n}{c\log^2 n}$ and polynomial running time $O\left(n^{\frac{\pi\sqrt{6}}{3}\sqrt{c}\log e\sqrt{\frac{\alpha(1-\alpha)}{\beta-\alpha}}}\log^3 n + n^2\right)$ for any $c > 0$. The algorithm works by trying all sorted ticket assignments with total number at most L, to be defined below, and if no valid assignment was found, proceeding to using faster Swiper as fallback.

By [39], Swiper always finds a solution with cost at most $B = \frac{\alpha(1-\alpha)}{\beta-\alpha}n + 1$. By setting $L = \left\lceil \frac{Bc\log^2 n}{n} \right\rceil - 1$, the approximation factor of the algorithm is at most $\frac{B}{L+1} \leq \frac{n}{c\log^2 n}$.

[1] https://en.wikipedia.org/wiki/Integer_partition

Similarly to subsection 8.1, the running time of the bruteforce is at most

$$O\left(n + L^{3/2}\exp\left(\frac{\pi\sqrt{6}}{3}\sqrt{L}\right)\right) = O\left(n + \log^3 n \cdot \exp\left(\frac{\pi\sqrt{6}}{3}\sqrt{c}\log n\sqrt{\frac{B}{n}}\right)\right) =$$
$$O\left(n + n^{\frac{\pi\sqrt{6}}{3}\sqrt{c}\log e\sqrt{\frac{B}{n}}} \cdot \log^3 n\right) = O\left(n + n^{\frac{\pi\sqrt{6}}{3}\sqrt{c}\log e\sqrt{\frac{\alpha(1-\alpha)}{\beta-\alpha}}} \log^3 n\right),$$

while the running time of faster Swiper is $O(n^2)$.

9 Numerical evaluation

We implemented algorithms from this paper and from the literature in the Rust programming language and evaluated them based on the total number of tickets in the output and the running time. The stake distributions of Algorand, Aptos, Filecoin and Tezos blockchains were used as inputs, borrowed from the original Swiper implementation's git repository[2]; they were used for evaluations in the Swiper paper [38, 39].

The implemented algorithms from our work are faster Swiper (subsection 5.1), extended Swiper (subsection 5.2) and the exact algorithm from subsection 8.1. The implemented algorithms from the literature are Swiper and gcdWR. Even though the original implementation of Swiper is publicly available, it is written in Python, a much slower programming language. To make the comparison fair, we reimplemented Swiper in Rust with the two implementations having identical output and almost identical logic; the only difference is that we did not implement the quasilinear time approximate knapsack solver that Swiper runs before invoking the full solver, but [39] states that this optimization improves the running time only by a factor of 3 and only when the number of tickets in the output is large. We include both implementations of Swiper in our evaluations. The original implementation in Python was slightly modified to solve the $(\leq \alpha, < \beta)$-weight reduction problem instead of the variant where the inequality involving α is strict. The LP based algorithm from Appendix A was not implemented due to its impractically large time complexity and the unavailability of an ellipsoid method implementation with a separation oracle, and the general technical complexity of such an implementation.

All our implementations, as well as the original Swiper code in Python, are single-threaded. We assume sorted weights in the input, and thus our faster Swiper implementation has time complexity $O(R^2)$ as opposed to $O(n+R^2)$ in the full version of the algorithm, with a similar difference for the extended Swiper and exact schemes. Even though, we didn't implement subsection 5.3 to handle unsorted inputs, we believe it does not significantly affect the measured running times since the stake distributions used all have less than 43000 entries. Our code is publicly available[3].

We ran the tests on AMD Threadripper PRO 7955WX. Tables 1, 2, 3 and 4 have two values for each evaluation: the total number of tickets and the running time. The exact algorithm was modified to search for a solution with the total number tickets up to 143. In case a solution wasn't found, the tables have a blank cell for the total number of tickets and the running time reported is the time duration before the search terminated.

[2] https://github.com/DCL-TelecomParis/swiper
[3] https://github.com/tolikzinovyev/weight-reduction

9.1 Discussion

In Table 1 we indeed see that our faster Swiper algorithm is faster than the original Swiper. The speedup on Algorand's stake distribution ranges from 17x when the cost of the output is large to 13000x when the output cost is small. The huge efficiency gap in the small output case can be attributed to expensive rational number arithmetic when linearly scaling weights down in Swiper (time complexity $O(n \log n)$), which can dominate even the total cost of knapsack (time complexity $O(n^2 \log n)$). In contrast, our faster Swiper implementation does only $O(R \log R)$ rational number operations (inside gen.next() – see subsection 5.1). The cost of the output is comparable between Swiper and faster Swiper, but not exactly the same, as one or the other can get stuck in "local minimums" depending on luck.

Next, we observe that our extended Swiper algorithm sometimes significantly improves the quality of the output over Swiper. For instance, for Aptos' stake distribution (Table 2), Swiper outputs 85 tickets for parameters $\alpha = 1/4, \beta = 1/3$ whereas extended Swiper outputs 58; for $\alpha = 3/10, \beta = 1/3$, Swiper and faster Swiper output 346 tickets whereas extended Swiper outputs 277. It is guaranteed that the output of extended Swiper is never worse than that of Swiper and faster Swiper; moreover, in our measurements it is always better than that of gcdWR and is always within 6% of the optimal solution (based on the exact algorithm when it is feasible). Where the gap between α and β is not too small, the running time of extended Swiper is not significantly larger than even that of faster Swiper.

Finally, it is worth noting that while the exact algorithm has sub-exponential running time, it can nonetheless be practical when the optimal solution is small (e.g., in the case of Aptos). It could thus be used in conjunction with a fallback scheme when the optimal solution turns out larger than a threshold.

10 Conclusion

In this paper we made further progress in solving the problem of deterministic weight reduction. On the practical side, faster Swiper is a very efficient algorithm with time complexity $O(n+R^2)$ that outputs solutions comparable to the optimal on real-world inputs while extended Swiper outputs solutions extremely close to optimal in a reasonable time $O(n + R^{2.5})$. Both would be great candidates for deployment in, e.g., proof-of-stake blockchains.

On the theoretical side, we proved that all known algorithms based on linear scaling suffer from $\Omega(n)$ approximation ratio in the worst case. We presented the first polynomial time algorithm with approximation factor $n/(c \log^2 n)$, for any $c > 0$, and the first exact algorithm with sub-exponential running time $O\left(n + R^{3/2} e^{C\sqrt{R}}\right)$. Given that R is often much smaller than n, our exact algorithm can also be practical for production or research purposes. Finally, our linear programming based algorithm could be useful in the context of future research.

It could be interesting to try integrating a quasilinear time approximate knapsack solver, as done in the quasilinear mode Swiper [39], to decrease the exponent of R by one in the time complexity of faster Swiper, extended Swiper and the exact scheme. Additionally, we would like to know if there exists an algorithm with approximation factor $o(n/\log^2 n)$. The answer is likely yes, but more exploration is required.

Finally, it would be good to investigate the related variants of committee selection where we require the weaker ("randomized") security from Equation 1 or where the goal is to minimize the number of distinct committee members (see section 2). [28] makes progress in the "randomized" setting presenting schemes performing very well in practice, but, unfortunately, nothing is said about their approximation factor. It would be interesting to prove that a fully deterministic solution can be optimal when the weaker security is required.

References

1 Ittai Abraham, Eli Chouatt, Ivan Damgård, Yossi Gilad, Gilad Stern, and Sophia Yakoubov. Asynchronous algorand: Reaching agreement with near linear communication and constant expected time. Cryptology ePrint Archive, Report 2025/303, 2025. URL: https://eprint.iacr.org/2025/303.

2 Georg Anegg, Haris Angelidakis, Adam Kurpisz, and Rico Zenklusen. A technique for obtaining true approximations for k-center with covering constraints. *Math. Program.*, 192(1–2):3–27, March 2022. doi:10.1007/s10107-021-01645-y.

3 Amos Beimel and Enav Weinreb. Monotone circuits for monotone weighted threshold functions. *Information Processing Letters*, 97(1):12–18, 2006. doi:10.1016/j.ipl.2005.09.008.

4 Fabrice Benhamouda, Shai Halevi, and Lev Stambler. Weighted secret sharing from wiretap channels. In Kai-Min Chung, editor, *ITC 2023*, volume 267 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl, June 2023. doi:10.4230/LIPIcs.ITC.2023.8.

5 Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 742–762. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_36.

6 Elette Boyle, Ran Cohen, Deepesh Data, and Pavel Hubácek. Must the communication graph of MPC protocols be an expander? *Journal of Cryptology*, 36(3):20, July 2023. doi:10.1007/s00145-023-09460-8.

7 Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the $O(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. *Journal of Cryptology*, 37(1):2, January 2024. doi:10.1007/s00145-023-09484-0.

8 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 356–376. Springer, Berlin, Heidelberg, March 2013. doi:10.1007/978-3-642-36594-2_21.

9 Gabriel Bracha. An $O(\lg n)$ expected rounds randomized byzantine generals protocol. In *17th ACM STOC*, pages 316–326. ACM Press, May 1985. doi:10.1145/22145.22180.

10 Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In Panagiota Fatourou and Gadi Taubenfeld, editors, *32nd ACM PODC*, pages 57–64. ACM, July 2013. doi:10.1145/2484239.2484243.

11 Ran Canetti, Sebastian Kolby, Divya Ravi, Eduardo Soria-Vazquez, and Sophia Yakoubov. Taming adaptivity in YOSO protocols: The modular way. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part II*, volume 14370 of *LNCS*, pages 33–62. Springer, Cham, November / December 2023. doi:10.1007/978-3-031-48618-0_2.

12 John F. Canny and Stephen Sorkin. Practical large-scale distributed key generation. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 138–152. Springer, Berlin, Heidelberg, May 2004. doi:10.1007/978-3-540-24676-3_9.

13 Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In David B. Shmoys, editor, *11th SODA*, pages 106–115. ACM-SIAM, January 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338241.

14 Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 651–680. Springer, Cham, December 2022. doi:10.1007/978-3-031-22963-3_22.

15 Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. In Markulf Kohlweiss, Roberto Di Pietro, and Alastair R. Beresford, editors, *CANS 2024, Part I*, volume 14905 of *LNCS*, pages 239–263. Springer, Singapore, September 2024. doi:10.1007/978-981-97-8013-6_11.

16 Pyrros Chaidos, Aggelos Kiayias, Leonid Reyzin, and Anatoliy Zinovyev. Approximate lower bound arguments. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part IV*, volume 14654 of *LNCS*, pages 55–84. Springer, Cham, May 2024. doi:10.1007/978-3-031-58737-5_3.

17 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019. In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part I. doi:10.1016/j.tcs.2019.02.001.

18 Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. From fairness to full security in multiparty computation. *Journal of Cryptology*, 35(1):4, January 2022. doi:10.1007/s00145-021-09415-x.

19 Varsha Dani, Valerie King, Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Secure multi-party computation in large networks. *Distrib. Comput.*, 30(3):193–229, June 2017. doi:10.1007/s00446-016-0284-9.

20 Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 356–370. ACM Press, November 2023. doi:10.1145/3576915.3623096.

21 Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. Distributed randomness using weighted VRFs. Cryptology ePrint Archive, Report 2024/198, 2024. URL: https://eprint.iacr.org/2024/198.

22 Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. Distributed randomness using weighted VUFs. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VII*, volume 15607 of *LNCS*, pages 314–344. Springer, Cham, May 2025. doi:10.1007/978-3-031-91098-2_12.

23 Hanwen Feng, Tiancheng Mai, and Qiang Tang. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2636–2650. ACM Press, October 2024. doi:10.1145/3658644.3690253.

24 Nils Fleischhacker, Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Jackpot: Non-interactive aggregatable lotteries. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 365–397. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0938-3_12.

25 G.N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104(2):197–214, 1993. doi:10.1006/inco.1993.1030.

26 Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: MPC, encryption and signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 295–327. Springer, Cham, August 2023. doi:10.1007/978-3-031-38557-5_10.

27 Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy*, pages 3034–3052. IEEE Computer Society Press, May 2024. doi:10.1109/SP54263.2024.00057.

28 Peter Gazi, Aggelos Kiayias, and Alexander Russell. Fait accompli committee selection: Improving the size-security tradeoff of stake-based committees. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 845–858. ACM Press, November 2023. doi:10.1145/3576915.3623194.

29 Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84245-1_3.

30 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on*

Operating Systems Principles, SOSP '17, pages 51–68, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3132747.3132757.

31 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-78240-4.

32 D. Ellis Hershkowitz, Nathan Klein, and Rico Zenklusen. Ghost value augmentation for k-edge-connectivity. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 1853–1864. ACM Press, June 2024. doi:10.1145/3618260.3649715.

33 Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Trans. Algorithms*, 6(4), September 2010. doi:10.1145/1824777.1824788.

34 Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In Marcos K. Aguilera, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and Romit Roy Choudhury, editors, *Distributed Computing and Networking*, pages 203–214, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-17679-1_18.

35 Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In Idit Keidar, editor, *Distributed Computing*, pages 464–478, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-04355-0_47.

36 Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4), July 2011. doi:10.1145/1989727.1989732.

37 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *17th SODA*, pages 990–999. ACM-SIAM, January 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109667.

38 Andrei Tonkikh and Luciano Freitas de Souza. Swiper: a new paradigm for efficient weighted distributed protocols. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM PODC*, pages 283–294. ACM, June 2024. doi:10.1145/3662158.3662799.

39 Andrei Tonkikh and Luciano Freitas. Swiper: a new paradigm for efficient weighted distributed protocols. Cryptology ePrint Archive, Report 2023/1164, 2023. URL: https://eprint.iacr.org/2023/1164.

40 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, USA, 1st edition, 2011. doi:10.1017/CBO9780511921735.

41 Anatoliy Zinovyev. Weight reduction. Software, DARPA, swhId: swh:1:dir:a48c33feba04bc98c7013c4c88b174f5e461a392 (visited on 2025-09-05). URL: https://github.com/tolikzinovyev/weight-reduction/tree/f1b5ce44aab9e4809bff67c5c6bff4acf1f82cb1, doi:10.4230/artifacts.24587.

42 Anatoliy Zinovyev. Weight reduction in distributed protocols: new algorithms and analysis. Cryptology ePrint Archive, Paper 2025/1076, 2025. URL: https://eprint.iacr.org/2025/1076.

A Linear programming based polynomial time approximation

In this section we show that for any constant $0 < \delta < 1 - \alpha/\beta$ there exists a polynomial time approximation algorithm that returns a solution with cost at most

$$\left\lfloor \frac{\mathsf{OPT}^{T_A}_{\leq\alpha,<(1-\delta)\beta}}{\beta} \right\rfloor + 1 = \left\lfloor (1-\delta) \cdot \frac{\mathsf{OPT}^{T_A}_{\leq\alpha,<(1-\delta)\beta}}{(1-\delta)\beta} \right\rfloor + 1 \leq \left\lfloor (1-\delta) \cdot \mathsf{OPT}_{\leq\alpha,<(1-\delta)\beta} \right\rfloor + 1 \leq \mathsf{OPT}_{\leq\alpha,<(1-\delta)\beta}, \tag{4}$$

where the first inequality is due to Theorem 4.

Table 1 Evaluation on the Algorand stake distribution.

	$\beta = 1/3$			$\beta = 1/2$			
	$\alpha = 0.2$	$\alpha = 0.25$	$\alpha = 0.3$	$\alpha = 0.3$	$\alpha = 0.35$	$\alpha = 0.4$	$\alpha = 0.45$
Swiper (Python)	235	745	22393	203	383	1203	17591
	4.75s	5.14s	6.25s	4.32s	5.15s	4.74s	5.70s
Swiper (Rust)	235	745	22393	203	383	1203	17591
	1.10s	1.21s	1.77s	1.03s	1.14s	1.18s	1.64s
faster Swiper	232	745	22660	203	387	1205	17949
	84.2µs	516µs	63.4ms	108µs	272µs	1.90ms	97.2ms
extended Swiper	232	745	22384	201	383	1171	17581
	98.9µs	849µs	3.48s	136µs	497µs	4.56ms	4.52s
gcdWR	1180	6237	178906	223	407	1180	17648
	3.09ms	2.85ms	2.42ms	3.29ms	3.27ms	3.11ms	2.66ms
exact							
	6.26h	4.50h	4.45h	8.72h	8.74h	8.70h	9.25h

Table 2 Evaluation on the Aptos stake distribution.

	$\beta = 1/3$			$\beta = 1/2$			
	$\alpha = 0.2$	$\alpha = 0.25$	$\alpha = 0.3$	$\alpha = 0.3$	$\alpha = 0.35$	$\alpha = 0.4$	$\alpha = 0.45$
Swiper (Python)	22	85	346	23	29	95	279
	5.37ms	7.32ms	6.94ms	4.87ms	6.19ms	6.76ms	7.06ms
Swiper (Rust)	22	85	346	23	29	95	279
	1.61ms	2.07ms	1.97ms	1.40ms	1.87ms	1.80ms	1.98ms
faster Swiper	22	58	346	23	31	95	279
	7.93µs	13.7µs	120µs	8.02µs	8.34µs	32.8µs	160µs
extended Swiper	22	58	277	23	29	95	241
	5.17µs	13.7µs	102µs	5.22µs	8.14µs	32.6µs	147µs
gcdWR	138	246	579	34	34	138	321
	7.96µs	7.85µs	7.86µs	8.46µs	8.68µs	7.98µs	7.98µs
exact	22	55		23	29	91	
	140µs	130ms	4.46h	227µs	1.14ms	47.5s	8.75h

Table 3 Evaluation on the Filecoin stake distribution.

	$\beta = 1/3$			$\beta = 1/2$			
	$\alpha = 0.2$	$\alpha = 0.25$	$\alpha = 0.3$	$\alpha = 0.3$	$\alpha = 0.35$	$\alpha = 0.4$	$\alpha = 0.45$
Swiper (Python)	1282	3091	10288	1149	1835	3533	8133
	281ms	324ms	388ms	287ms	301ms	322ms	430ms
Swiper (Rust)	1282	3091	10288	1149	1835	3533	8133
	29.6ms	40.2ms	98.2ms	31.7ms	35.8ms	49.8ms	129ms
faster Swiper	1282	3088	10288	1149	1837	3533	8139
	2.72ms	8.21ms	65.1ms	3.46ms	4.52ms	11.8ms	37.9ms
extended Swiper	1279	3088	10282	1147	1835	3527	8133
	3.52ms	23.4ms	305ms	5.15ms	18.2ms	67.5ms	399ms
gcdWR	3700	6516	17425	1289	1882	3700	8190
	166µs	162µs	158µs	169µs	170µs	166µs	162µs
exact							
	4.49h	4.48h	4.49h	8.51h	9.05h	8.56h	8.57h

Table 4 Evaluation on the Tezos stake distribution.

	$\beta = 1/3$			$\beta = 1/2$			
	$\alpha = 0.2$	$\alpha = 0.25$	$\alpha = 0.3$	$\alpha = 0.3$	$\alpha = 0.35$	$\alpha = 0.4$	$\alpha = 0.45$
Swiper (Python)	49	133	589	37	75	143	455
	28.5ms	28.3ms	31.3ms	20.1ms	29.2ms	25.8ms	28.9ms
Swiper (Rust)	49	133	589	37	75	143	455
	3.35ms	3.79ms	4.59ms	2.72ms	3.97ms	3.43ms	4.26ms
faster Swiper	49	133	589	41	77	145	455
	11.9µs	55.0µs	383µs	12.9µs	29.7µs	66.8µs	188µs
extended Swiper	37	127	523	35	75	143	427
	6.25µs	36.4µs	533µs	6.77µs	21.8µs	54.0µs	597µs
gcdWR	148	333	1331	46	87	148	470
	13.7µs	12.7µs	11.3µs	14.7µs	13.7µs	13.7µs	12.2µs
exact	37	127		35	75	139	
	6.44ms	1.05h	5.07h	5.36ms	6.64s	6.33h	8.62h

The algorithm is based on the standard approach of representing an optimization problem as an integer linear program, relaxing it to allow fractional solutions, solving the relaxed LP in polynomial time and rounding the solution to an integer vector in a problem specific manner [40]. We will use the ellipsoid method to solve an LP with exponentially many constraints. In this mode of operation, we are required to implement a "separation oracle" that the ellipsoid method will invoke. On input some vector $\vec{x} \in \mathbb{Q}^n$ in the multidimensional space, the separation oracle either reports that $\vec{x}$ satisfies all LP constraints or returns some violated constraint. Whenever the separation oracle is implemented by a polynomial time algorithm, the ellipsoid method also runs in polynomial time; in particular, its running time is independent of the number of LP constraints. See [40, section 4.3] for an overview of the ellipsoid method with a separation oracle.

For our problem, however, we face an issue that we cannot implement a polynomial time separation oracle that calculates the prescribed output exactly. The input will be a vector $\vec{t} \in \mathbb{Q}^n$ and we need to check if it satisfies the criteria of a valid ticket assignment. While we can test a ticket assignment $(t_1, ..., t_n) \in \mathbb{Z}^n_{\geq 0}$ for validity in time $O\big(n \cdot \sum_{i=1}^n t_i\big)$, we do not know how to efficiently do so for a fractional $\vec{t}$. We first present a sketch of our weight reduction algorithm that uses an inefficient separation oracle and in subsection A.2 we show how to get around this issue.

A.1 Sketch

Assume $w_1 \leq w_2 \leq ... \leq w_n$ and define $k = \lceil 1/\delta \rceil$. We first efficiently bruteforce the set of indices i with $t_i = g$ for each $0 \leq g \leq k-2$, having Theorem 2 in mind. Now consider the following satisfiability LP with some constant T, some t_i fixed by the "guess" and the other t_i constrained to be at least $k-1$.

$$\begin{aligned}
&\text{minimize} && 0 && \\
&\text{subject to} && \sum_{i=1}^{n} t_i = T && \\
& && \sum_{i \in I} t_i \leq \big\lfloor (1-1/k)\beta T \big\rfloor && \forall I \subseteq [n] \text{ s.t. } \sum_{i \in I} w_i \leq \alpha \sum_{i=1}^{n} w_i \\
& && t_i = g_i && \forall i : 1 \leq i < j \\
& && t_i \geq k-1 && \forall i : j \leq i \leq n
\end{aligned} \tag{5}$$

For each $T \in \{1, 2, ..., B\}$, where $B = O(n)$ is a bound on $\mathsf{OPT}_{\leq \alpha, < (1-1/k)\beta}$ from [39], if there is a solution $(t'_1, ..., t'_n)$ to this LP, round it down to get $(t_1, ..., t_n)$, let $T_{\mathsf{A}} = \big\lfloor (1-1/k)\beta T \big\rfloor$, reduce all t_i as much as possible as long as $T_{\mathsf{A}} < \beta \sum_{i=1}^n t_i$ to get $(t^+_1, ..., t^+_n)$ and record $(t^+_1, ..., t^+_n)$ as the best result for the current guess $(g_1, ..., g_{j-1})$. At the end of the algorithm, return the smallest result across all guesses.

We will now prove the bound in Equation 4 on the cost of the returned solution. By Theorem 2, fix an optimal solution $(t^*_1, ..., t^*_n)$ with $t^*_1 \leq t^*_2 \leq ... \leq t^*_n$ to the $(\leq \alpha, \leq (1-1/k)\beta)$-weight reduction problem. By definition, $\sum_{i=1}^n t^*_i = \mathsf{OPT}_{\leq \alpha, \leq (1-1/k)\beta} \leq \mathsf{OPT}_{\leq \alpha, < (1-1/k)\beta}$. Let $(g_1, ..., g_{j-1}) \in \{0, ..., k-2\}^{j-1}$ be the guess such that $(g_1, ..., g_{j-1}) = (t^*_1, ..., t^*_{j-1})$ and $t^*_i \geq k-1$ for $i \geq j$.

For this guess, the algorithm will terminate the search on some T at most $\mathsf{OPT}_{\leq \alpha, \leq (1-1/k)\beta}$. This is because if $T = \mathsf{OPT}_{\leq \alpha, \leq (1-1/k)\beta}$, then $(t^*_1, ..., t^*_n)$ is a valid solution to the above LP. Therefore, the algorithm will find a solution $(t'_1, ..., t'_n)$ to the LP with $\sum_{i=1}^n t'_i = T \leq \mathsf{OPT}_{\leq \alpha, \leq (1-1/k)\beta}$.

For any $I \subseteq [n]$ that the adversary is allowed to corrupt,

$$\sum_{i \in I} t_i \leq \sum_{i \in I} t'_i \leq \lfloor (1-1/k)\beta T \rfloor = T_{\mathsf{A}} = \left\lfloor (1-1/k)\beta \sum_{i=1}^{n} t'_i \right\rfloor \leq (1-1/k)\beta \sum_{i=1}^{n} t'_i < \beta \sum_{i=1}^{n} t_i,$$

where the last inequality holds by the assumption that every t'_i is either an integer or at least $k-1$, and that at least one t'_i is non-zero. Therefore, $(t_1^+, ..., t_n^+)$ is a valid solution to the $(\leq \alpha, < \beta)$-weight reduction problem. Moreover, $T_{\mathsf{A}} \leq \lfloor (1-1/k)\beta \cdot \mathsf{OPT}_{\leq \alpha, \leq (1-1/k)\beta} \rfloor = \mathsf{OPT}^{T_{\mathsf{A}}}_{\leq \alpha, \leq (1-1/k)\beta} \leq \mathsf{OPT}^{T_{\mathsf{A}}}_{\leq \alpha, < (1-\delta)\beta}$, where the equality is due to Theorem 5. Hence,

$$\sum_{i=1}^{n} t_i^+ = \left\lfloor \frac{T_{\mathsf{A}}}{\beta} \right\rfloor + 1 \leq \left\lfloor \frac{\mathsf{OPT}^{T_{\mathsf{A}}}_{\leq \alpha, < (1-\delta)\beta}}{\beta} \right\rfloor + 1.$$

A.2 Full algorithm

We now show how to circumvent the issue that we cannot construct a polynomial time separation oracle for the LP in Equation 5. We utilize the round-or-cut framework [13, 2] where at each iteration of the ellipsoid method, we either produce a separating hyperplane between the candidate feasible solution $\vec{t} \in \mathbb{Q}^n$ and the polytope or return another point $\vec{t'}$ that is slightly different from $\vec{t}$ but that will suffice for our weight reduction algorithm.

Configure the ellipsoid method with the LP in Equation 5 (see details in [31]) and implement the separation oracle as follows. On input $(t_1, ..., t_n) \in \mathbb{Q}^n$, check the constraints in lines 1, 3 and 4 in Equation 5 as usual; instead of the constraints in line 2, check relaxed constraints

$$\sum_{i \in I} \lfloor t_i \rfloor \leq \lfloor (1-1/k)\beta T \rfloor \quad \forall I \subseteq [n] \text{ s.t. } \sum_{i \in I} w_i \leq \alpha \sum_{i=1}^{n} w_i.$$

using knapsack in polynomial time. If some check failed, we return one of constraints from the original LP in Equation 5 as a separating hyperplane; otherwise, we terminate the search and return $(t_1, ..., t_n)$ as the solution.

The result of this search will be either a point $(t_1, ..., t_n)$ that satisfies

$$\begin{aligned}
&\sum_{i=1}^{n} t_i = T \\
&\sum_{i \in I} \lfloor t_i \rfloor \leq \lfloor (1-1/k)\beta T \rfloor \quad && \forall I \subseteq [n] \text{ s.t. } \sum_{i \in I} w_i \leq \alpha \sum_{i=1}^{n} w_i \\
&t_i = g_i && \forall i : 1 \leq i < j \\
&t_i \geq k-1 && \forall i : j \leq i \leq n
\end{aligned}$$

or an error certifying that the LP in Equation 5 has no solution. The rest of the algorithm and the proof of correctness remains the same.

A.3 Time complexity

We briefly argue that the algorithm runs in polynomial time. Using Theorem 2, bruteforcing the set of indices i with $t_i = j$ for each $0 \leq j \leq k-2$ takes time $O(n^{k-1})$. Bruteforcing T adds another factor of $O(n)$ by construction. Finally, solving the LP takes polynomial time.

Despite this, the algorithm is completely impractical since for reasonable parameters, the running time would be $\Omega(n^{10})$.

A.4 Approximation factor discussion

We have already proved that the algorithm finds a solution with cost at most

$$\left\lfloor \frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<(1-\delta)\beta}}{\beta} \right\rfloor + 1.$$

Using Theorem 4, the approximation factor for the total number of tickets is at most

$$\frac{\left\lfloor \frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<(1-\delta)\beta}}{\beta} \right\rfloor + 1}{\left\lfloor \frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}}{\beta} \right\rfloor + 1} < \frac{\frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<(1-\delta)\beta}}{\beta} + 1}{\frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}}{\beta}} = \frac{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<(1-\delta)\beta}}{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}} + \frac{\beta}{\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}}.$$

While $\beta/\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}$ can be made as small as $1/\log^2 n$ by subsection 8.2, in the full version of our paper [42, Appendix B] we show that, unfortunately, $\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<(1-\delta)\beta}/\mathsf{OPT}^{T_\mathsf{A}}_{\leq\alpha,<\beta}$ can be $\Omega(n)$ in the worst case.

B Proofs

Proof of Theorem 3. Let $(t_1, ..., t_n)$ be any solution and take any i, j such that $w_i = w_j$ and $t_i + 2 \leq t_j$. Define $(t'_1, ..., t'_n)$ such that $t'_i = t_i + 1$, $t'_j = t_j - 1$ and $t'_k = t_k$ for all other k. We will show that this is also a valid solution.

Let $I \subseteq [n]$ such that $\sum_{k\in I} w_k \leq \alpha \sum_{k=1}^n w_k$ and consider the following cases.

- $i \notin I$, $j \notin I$: $\sum_{k\in I} t'_k = \sum_{k\in I} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$;
- $i \in I$, $j \in I$: $\sum_{k\in I} t'_k = \sum_{k\in I} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$;
- $i \notin I$, $j \in I$: $\sum_{k\in I} t'_k < \sum_{k\in I} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$;
- $i \in I$, $j \notin I$: let $I' = I \setminus \{i\} \cup \{j\}$, $\sum_{k\in I} t'_k < \sum_{k\in I'} t_k < \beta \sum_{k=1}^n t_k = \beta \sum_{k=1}^n t'_k$, where the inequality holds since $\sum_{k\in I'} w_k = \sum_{k\in I} w_k \leq \alpha \sum_{k=1}^n w_k$.

Hence, $(t'_1, ..., t'_n)$ is a valid assignment.

By Theorem 2, take an optimal solution $(t_1, ..., t_n)$ with $t_1 \leq t_2 \leq ... \leq t_n$. One can see that we can iteratively achieve $|t_i - t_j| \leq 1$ for any block of indices $\{l, l+1, ..., r\}$ with equal weight, while preserving the sortedness of the whole array. ◀

Proof of Theorem 6.

$$\sum_{i=1}^{k} i e^{c\sqrt{i}} \leq \int_1^{k+1} x e^{c\sqrt{x}}\, dx\ [=]$$

letting $y = c\sqrt{x}$,

$$[=] \int_{y=c}^{y=c\sqrt{k+1}} \left(\frac{y}{c}\right)^2 e^y\, d\left(\left(\frac{y}{c}\right)^2\right) = \int_c^{c\sqrt{k+1}} \left(\frac{y}{c}\right)^2 e^y \frac{2y}{c^2}\, dy =$$

$$\frac{2}{c^4} \int_c^{c\sqrt{k+1}} y^3 e^y\, dy = \frac{2}{c^4} \int_{y=c}^{y=c\sqrt{k+1}} y^3\, d(e^y)\ [=]$$

letting $z = e^y$, for sufficiently large k,

$$[=] \frac{2}{c^4} \int_{e^c}^{e^{c\sqrt{k+1}}} \ln^3 z\, dz = \frac{2}{c^4} \Big(z\big(\ln^3 z - 3\ln^2 z + 6\ln z - 6\big) \Big) \Big|_{e^c}^{e^{c\sqrt{k+1}}} \leq$$

$$\frac{2}{c^4} \cdot e^{c\sqrt{k+1}} \cdot \Big(c\sqrt{k+1}\Big)^3 = \frac{2}{c} \cdot (k+1)^{3/2} \cdot e^{c\sqrt{k+1}} = O\Big(k^{3/2} e^{c\sqrt{k}}\Big).$$

◀

Brief Announcement: Incrementally Verifiable Distributed Computation

Eden Aldema Tshuva
Tel Aviv University, Israel

Rotem Oshman
Tel Aviv University, Israel

Abstract

Incrementally verifiable computation (IVC) is a cryptographic scheme that allows a prover to certify the correctness of a long or ongoing computation in an incremental manner, by repeatedly updating a proof certifying the computation so far. Updating the proof does not require access to the entire trace of the computation, which makes the IVC prover memory efficient.

In this work we construct incrementally verifiable *distributed* computation, which allows a distributed algorithm to efficiently certify its own execution using low memory and communication overhead. Our primary motivation is massively-parallel computation (MPC), where memory efficiency is make-or-break: the machines participating in an MPC algorithm usually cannot store the entire trace of their computation. Thus, certifying MPC algorithms essentially *requires* distributed IVC.

At the heart of this work is a new abstraction, *updatable batch arguments for* NP (UpBARGs), which we define and construct. Standard BARGs allow one to prove a batch of k NP-statements using a proof whose length barely grows with k; however, the statements and their witnesses must all be known in advance. In contrast, UpBARGs support adding statements and witnesses on the fly, making them a flexible tool for constructing IVC across different computational models. We use UpBARGs to construct IVC for streaming algorithms, for MPC algorithms, and for PRAM algorithms in the exclusive-read exclusive-write (EREW) model.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases Incrementally verifiable computation, massively parallel computation, streaming, parallel RAM, batch arguments, SNARG

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.44

Funding *Eden Aldema Tshuva*: Research supported by the Israeli Science Foundation, Grant No. 2338/23, and by AFOSR award FA9550-24-1-0156.
Rotem Oshman: Research funded by the Israel Science Foundation, Grant No. 2801/20, and by AFOSR award FA9550-24-1-0156.

1 Introduction

Incrementally verifiable computation (IVC), introduced by Valiant [15], enables a prover to generate a succinct proof that a long computation has been executed correctly, and update the proof incrementally as the computation progresses: given a proof that the first t steps have been executed correctly (for some t), the prover can update it to obtain a proof that the first $t+1$ steps have been executed correctly, without having to store in memory the entire trace of the computation so far. This makes IVC a natural fit for scenarios where long computations must be executed reliably over time: for example, resuming a paused computation from an intermediate verified state, or continuously auditing the correctness of outsourced computation performed by an untrusted server or cloud.

The incremental structure of IVC is particularly well-suited to computations that are *distributed* or *reactive* in nature: where inputs arrive over time, or are jointly held by multiple parties that collaborate to carry out the computation. In this work we extend the scope of


© Eden Aldema Tshuva and Rotem Oshman;
licensed under Creative Commons License CC-BY 4.0



IVC to encompass such distributed and reactive models of computation, bringing the benefits of incremental verification to a setting that arguably stands to gain from it the most. We construct IVC for massively parallel computation (MPC), streaming algorithms and PRAM. Our main technical tool is *updatable hash-and-*BARG *for* NP(UpBARGs), a generic building block that we believe is of independent interest: it allows us to construct IVC for models where we can prove the correctness of a *single* computation step.

IVC for reactive and distributed computation. An IVC scheme consists of a *prover* and a *verifier*. The prover is intended to run alongside the computation whose correctness it proves, so it is important for the prover to have low overhead in terms of space, and for distributed computations, also in terms of communication and rounds. At each point in time, the prover holds a proof that the computation has executed correctly *so far*, and as the computation proceeds, the prover updates the proof. The interface to the prover consists of a *proof update* procedure that takes a proof π_t reflecting some prefix of t computation steps, alongside the current state of the computation st_t, and returns a new proof π_{t+1} reflecting $t+1$ computation steps. The verifier is modeled as a sequential algorithm. Because the input to the system could be very large (e.g., in streaming or MPC algorithms), we do not assume that the verifier can store all of it, or even that the verifier can store a single global state of the system in its entirety. Instead, the verifier is presented with *hashes* h_0, h_t of two states $\mathsf{st}_0, \mathsf{st}_t$ of the computation (respectively), a number of steps t, and a proof π, and it checks whether π is a "convincing" proof that the computation indeed reaches st_t after t computation steps from st_0. We use the short-hand notation "$\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$" for this statement. The notions of "states" and "computation steps" depends on the exact computation model (e.g., it can mean synchronous rounds, or a single step of a process).

The fact that the verifier is given *hashes* h_0, h_t of the states $\mathsf{st}_0, \mathsf{st}_t$ instead of plaintext, presents some definitional subtleties, as one cannot reconstruct the original state from its hash value; technically, given h_0, h_t and t, the verifier does not even *know* what statement "$\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$" is being asserted. To resolve this we adopt definitions analogous to the one used for RAM delegation [11, 6], where the same issue arises. Informally, the distributed IVC schemes that we design have the following essential properties:

- Succinctness: the proof is short enough to store on a single machine.
- Incremental completeness: for any system states $\mathsf{st}_0, \mathsf{st}_t$ and st_{t+1} such that st_t transitions to st_{t+1} in one computation step, if the prover is given $\mathsf{st}_0, \mathsf{st}_t$ and a proof π_t such that the verifier accepts π_t as a proof for the statement $\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$, then the prover produces a proof π_{t+1} for the statement "$\mathsf{st}_0 \xrightarrow{t+1} \mathsf{st}_{t+1}$", which is also accepted by the verifier.[1]
- Computational soundness: no poly-size algorithm[2] can generate values h_0, h_t, h'_t, a step number t and two proofs π_t, π'_t, such that $h_t \neq h'_t$, but the verifier accepts both the proof π_t with h_0, h_t, t and the proof π'_t with h_0, h'_t, t. Intuitively, this means that the verifier cannot be convinced of two "contradictory statements", asserting that starting from some initial state (reflected by the hash h_0), after t computation steps the system ends up in two different states (reflected by the hashes $h_t \neq h'_t$).

The main and most challenging model we handle in this work is *massively parallel computation* (MPC) [7, 12], where n space-bounded machines collaborate to compute over an input initially partitioned between them. Proving the correctness of MPC computations

[1] Here, by "the verifier accepts π_t for $\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$" we mean it accepts (h_0, h_t, t, π) where h_0, h_t are the respective hash values of $\mathsf{st}_0, \mathsf{st}_t$.

[2] That is, an algorithm that can be represented as a circuit with a polynomial number of wires. This is at least as strong as probabilistic polynomial-time Turing machines.

almost *requires* IVC: non-incremental verification typically requires access to the entire trace of the computation at once, which in the MPC model no single machine can store. Thus, while efficient distributed provers have already been developed for other distributed models [1, 2], no such construction is known for MPC, even if we do not insist on IVC and are willing to settle for a non-incremental construction. Thus, our construction of IVC for MPC addresses not only the problem of incrementality but also the more basic problem of proving the correctness of massively parallel computation.

We consider two use cases. (1) Reliable cloud delegation: a user with input x outsources a heavy computation to a server farm or cloud provider, which returns both the output and a short correctness certificate. The user (a single weak machine) hashes x once, never rereads it, and verifies the result in polylogarithmic time in $|x|$. (2) Internal MPC verification: the network itself verifies correctness so far (e.g., to detect faulty machines). Our distributed IVC maintains a short, continuously updatable certificate that any machine can use to verify locally against its own input. This resembles proof labeling schemes [13] but with polylogarithmic, rather than overall linear proof growth, and with each machine verifying independently without communication.

Our framework for constructing IVC is modular and generic: we are able to construct IVC for practically any model of deterministic computation for which we can prove the correctness of a single computation step. We demonstrate this by constructing, in addition to the IVC for MPC discussed above, IVC for streaming algorithms, where a low-space single machine processes a stream of elements that arrive over time and cannot all be stored at once, and for PRAM algorithms in the exclusive-read exclusive-write model (EREW), which captures parallel algorithms in shared memory. While for streaming algorithms, proving the correctness of a single computation step is trivial (as the step itself is represented succinctly by the state transition and the stream element), for the PRAM and MPC models we use cryptographic proof systems called *succinct non-interactive arguments* (SNARGs): essentially, these are the non-incremental analog of IVC. Specifically, for PRAM we use the existing SNARG for PRAM of [9]. For MPC, no such construction is known, so we first construct a SNARG for one-round MPC, and then lift it into full-fledged IVC for MPC. Unlike some commonly used heuristically sound SNARKs (SNARGs of knowledge) and IVC [3, 10], all of our constructions are sound under *standard* cryptographic assumptions.

Our main building block: updatable hash-and-BARG for NP (UpBARGs). Proving that a computation $\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$ is correct boils down to proving the conjunction of t statements: "for each $i = 0, \ldots, t-1$, the system transitions from state st_i to state st_{i+1} in one step". In existing constructions of (non-incremental) SNARG from standard cryptographic assumptions [6, 16, 4], this is done using two cryptographic tools: *batch arguments (*BARG*s) for* NP and *hash families with local openings* (e.g., Merkle trees). Informally, a BARG for an NP-language $\mathcal{L}$ allows us to prove the conjunction of k statements, each of the form "$x_i \in \mathcal{L}$", using a proof whose length grows linearly with the length of a *single* NP-witness, and only polylogarithmically with the number of statements k. A hash family with local openings allows us to hash an n-sized vector v to a short hash value h, and in addition, given an index i, to produce a *short opening* ρ_i to the i-th index. Given only (h, i, v_i, ρ), one can verify that x_i is indeed the i-th entry of the vector whose hash is h.

In prior constructions, to prove (non-incrementally) that $\mathsf{st}_0 \xrightarrow{t} \mathsf{st}_t$, the prover hashes the vector $(\mathsf{st}_0, \ldots, \mathsf{st}_t)$ using a hash with local openings to obtain a hash value h. Then it constructs a BARG asserting that for each $i = 0, \ldots, t-1$, there exist states $\mathsf{st}_i, \mathsf{st}_{i+1}$ and openings ρ_i, ρ_{i+1} such that h opens to $\mathsf{st}_i, \mathsf{st}_{i+1}$ in positions $i, i+1$ (respectively), and st_i transitions to st_{i+1} in one step. Here, the states $\mathsf{st}_i, \mathsf{st}_{i+1}$ and the openings ρ_i, ρ_{i+1} serve as an NP-witness for the statement "the i-th step is reflected correctly in the hash h".

When incrementally proving the correctness of a reactive or distributed computation, we do not know in advance *what* statements the prover will need to prove, as the state that the system will reach depends on the future inputs or messages. Thus, we cannot lay out the entire computation trace in advance, hash it, and prepare a BARG proof. To overcome this, we design a scheme we call *updatable hash-and-*BARG (UpBARG), which imitates the above hash-and-BARG paradigm while allowing both the hash and the BARG to be constructed incrementally, together. This scheme enables us to update a hash value and a proof for a conjunction of NP-statements on-the-fly, without advance knowledge. To use our UpBARG to construct IVC, we must overcome another difficulty, which is that without having the entire trace in advance, we cannot compute the openings that serve as NP-witnesses even for the entries we already have. We show that we can compute the required openings *in hindsight*, using properties of tree-based hash families with local openings. Our techniques are inspired by the recent works on IVC for sequential deterministic computation [14, 8].

2 Brief Preliminaries

The common reference string model and computational hardness. Our work is set in the *common reference string* (CRS) model. In this model, all parties have access to a string that is sampled randomly by a trusted setup process, denoted by Gen, which takes a *security parameter* λ. (This can be viewed as public randomness.) The security parameter governs the computational resources that must be invested to break the security of the protocol: we say that a task that involves the CRS is *computationally hard* if given $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$, no poly-size (in λ) adversary can succeed in the task, except with *negligible probability* in λ.

Batch arguments for NP (BARGs). A batch argument (BARG) [5] allows a prover to succinctly prove the conjunction of k NP statements, where the size of the proof barely grows with k, and is similar to the size of one witness. The interface is as follows:

- $\mathcal{P}(\mathsf{crs}, \mathcal{M}, x_1, \ldots, x_k, w_1, \ldots, w_k) \to (\pi)$: takes a reference string crs, a machine $\mathcal{M}$, instances $x_1, \ldots, x_k$, and witnesses $w_1, \ldots, w_k$, and outputs a proof π.
- $\mathcal{V}(\mathsf{crs}, \mathcal{M}, \pi) \to b$: takes a reference string crs, a machine $\mathcal{M}$, and a proof π and outputs an acceptance bit b.

We say a BARG is *somewhere extractable* if it allows the extraction of *one witness* from a convincing proof π, as follows:

- The crs generation procedure Gen may be called in *trapdoor mode*. In trapdoor mode, Gen takes as additional input an index $i \in [k]$, called the *binding index*. It outputs a pair $(\mathsf{crs}, \mathsf{td}_i)$, where td is a *trapdoor* that can later be used to recover the i-th witness. In trapdoor mode, the Gen procedure has a property called *index hiding*: given crs (without td_i), it is computationally hard to *find* the binding index i.
- The BARG has the *somewhere argument of knowledge* property: There exists an efficient auxiliary *extraction* procedure, $\mathcal{E}(\mathsf{td}_i, \mathcal{M}, \pi) \to w_i$, which satisfies the following. Suppose we call Gen in trapdoor mode with a binding index i, and obtain $(\mathsf{crs}, \mathsf{td}_i)$. Given only crs, it is computationally hard to find a proof π that is accepted by the verifier, such that when we extract a witness w_i using $\mathcal{E}(\mathsf{td}_i, \mathcal{M}, \pi)$, we have $\mathcal{M}(x_i, w) \neq 1$.

3 Technical Overview

We give an overview of our distributed IVC constructions, with emphasis on the MPC model. Our framework for constructing IVC is generic and quite simple once we have (1) a SNARG for one-step computations in the model, and (2) an updatable hash-and-BARG scheme.

3.1 Defining and Using Updatable Hash-and-BARG

An updatable hash-and-BARG (UpBARG) is defined with respect to some hash family with local openings. It consists of three algorithms, $(\mathsf{Gen}, \mathcal{U}, \mathcal{V})$, where Gen is a standard CRS generation algorithm, and $\mathcal{U}, \mathcal{V}$ are update and verification algorithms, respectively. The syntax for the update and verification algorithms is as follows:

- $\mathcal{U}(\mathsf{hk}, \mathsf{crs}, \mathcal{M}, H, \Pi, x, w) \to (H^*, \Pi^*)$: takes as input a hash key hk, a common reference string crs, a Turing machine $\mathcal{M}$, a hash value H, a proof Π, and a new statement x and witness w, and outputs a hash value H and a new proof Π^*.
- $\mathcal{V}(\mathsf{hk}, \mathsf{crs}, \mathcal{M}, H, \Pi) \to b$: takes a hash key hk, a common reference string crs, a Turing machine $\mathcal{M}$, a hash value H, and proof Π, and outputs an acceptance bit $b \in \{0, 1\}$.

Our UpBARG satisfies the standard succinctness and index hiding properties of BARGs (see Section 2), but it has stronger completeness and argument of knowledge properties:

- *Incremental completeness:* if $\mathcal{V}$ accepts $(\mathsf{hk}, \mathsf{crs}, \mathcal{M}, H, \Pi)$, and $\mathcal{M}(x, w) = 1$, then it also accepts $(\mathsf{hk}, \mathsf{crs}, \mathcal{M}, H^*, \Pi^*)$, where $(H^*, \Pi^*) = \mathcal{U}(\mathsf{hk}, \mathsf{crs}, \mathcal{M}, H, \Pi, x, w)$.
- *Somewhere argument of knowledge:* upon using the trapdoor version of Gen, with binding index i, the extractor $\mathcal{E}$ extracts not only an instance-witness pair (x, w), but also an opening ρ proving that H indeed opens to x in the index i.

Supporting consistency checks. Our IVC requires the ability to claim that two extracted instances are *consistent* with each other, for a definition of "consistent" that is supplied by the user. To that end, we extend the syntax above to feed the update and verification algorithms additional inputs: a Turing machine $\mathcal{C}$ specifying the consistency check, and a "last instance" x^-. Our incremental completeness guarantee in this case holds whenever $\mathcal{C}(x^-, x) = 1$. The extractor of the somewhere argument of knowledge guarantee now additionally extracts a *previous instance* x^- and a respective opening, such that: (1) H opens to x^- in location $i - 1$ and this is proved by said opening, and (2) $\mathcal{C}(x^-, x) = 1$.

From UpBARG to IVC. We now aim to incrementally certify the correctness of execution of an ongoing computation, denote it by $\mathcal{M}$, where we wish the proof updating procedure to be done in the same model as $\mathcal{M}$. Given a SNARG for $\mathcal{M}$ which prover's run in the same model, the UpBARG provides the required building block fairly directly. Let $\mathsf{Gen}, \mathcal{P}, \mathcal{V}$ be a SNARG scheme in which we are able to prove (within the computation model) the correctness of one step $\mathsf{st}_i \to \mathsf{st}_{i+1}$. Let $\mathcal{M}_{\mathcal{V}}$ be the non-deterministic machine which verifies the SNARG proof. That is, on inputs (x, w), $\mathcal{M}_{\mathcal{V}}$ interprets x as $(\mathsf{st}_i, \mathsf{st}_{i+1})$, and w as a SNARG-proof π for the transition $\mathsf{st}_i \to \mathsf{st}_{i+1}$, then it applies the SNARG verifier and accepts or rejects accordingly.

The IVC prover runs alongside $\mathcal{M}$ and updates an UpBARG proof Π for the machine $\mathcal{M}_{\mathcal{V}}$, and whenever the algorithm transitions from state st to state st', it uses $\mathcal{P}$ to prove $\mathsf{st} \to \mathsf{st}'$ and then updates Π with the instance-witness pair $((\mathsf{st}, \mathsf{st}'), \pi)$. We use consistency checks to ensure that every two consecutive statements overlap on their last and first states, respectively, to prevent a situation where we correctly assert that "$\mathsf{st}_1 \to \mathsf{st}_2$" and then "$\mathsf{st}'_2 \to \mathsf{st}_3$", but $\mathsf{st}_2 \neq \mathsf{st}'_2$, so we do not truly prove that $\mathsf{st}_1 \xrightarrow{2} \mathsf{st}_3$. Consistency checks allow us to force $\mathsf{st}_2 = \mathsf{st}'_2$, so that the conjunction of all statements together asserts the correctness of the entire multi-step computation from the initial state to the current one.

3.2 SNARG for One-Round Massively Parallel Computation

Our generic "SNARG-to-IVC via UpBARG" method can lift a SNARG for one computation step in a computational model into a full IVC for that model. However, for the MPC model, no SNARG (even for a single round) was known prior to our work. We outline our construction of such a SNARG, using UpBARGs again.

One MPC round as a RAM program. We observe that one round of MPC on n machines can be described as a RAM program, which takes a vector of the machines' initial states $(\mathsf{st}(1), \ldots, \mathsf{st}(n))$, and outputs the machines' states in the end of the round $((\mathsf{st}'(1), \ldots, \mathsf{st}'(n))$. Each state $\mathsf{st}(i)$ or $\mathsf{st}'(i)$ is an s-bit vector, where s is the space bound of an individual machine. To construct a SNARG for one-round-MPC, we imitate a SNARG for the program above. However, while a RAM program is executed on a single machine that can access any place in its random-access memory, in the MPC model, the "memory" is partitioned across the n machines. Fortunately, existing RAM SNARGs have the property that constructing the proof does not require access to the entire memory configuration: it is enough to have a *hash with local openings* of the memory, and openings to the locations that are accessed by the algorithm in each step. Thus, to construct our desired RAM-SNARG-like proof for the MPC model, we need to go through the following three stages: (1) Obtain a hash of the state vector $(\mathsf{st}(1), \ldots \mathsf{st}(n))$; (2) For each machine $i \in [n]$, obtain openings of the above hash value to the messages $m_1^i, \ldots, m_k^i$ (where $k \leq s$) that machine i receives during the round[3]; and (3) Using the openings, construct a SNARG proof which proves the transition $(\mathsf{st}(1), \ldots, \mathsf{st}(n)) \to (\mathsf{st}'(1), \ldots, \mathsf{st}'(n))$. The main challenge in this scheme is that each stage must be implemented in the MPC network: we do not *truly* have random access to the state assignments, because these are actual states of individual machines. For all three stages, our solution is based on having the n machines simulate some process on a tree-structured network. This network is an s-ary tree with n leaves which represent the states of the n actual machines, and the rest of the nodes are virtual nodes, simulated by the n real machines.

Using a virtual tree-structured network. To obtain a hash of the state assignment $(\mathsf{st}(1), \ldots, \mathsf{st}(n))$, we aggregate the hash up the tree: the leaves send their state to their parents, and each inner node, upon receiving values from its s children, hashes the values it received and sends them up to its parent. The root of the tree then obtains a hash of the entire state assignment. In order for each machine $i \in [n]$ to obtain an opening to its state $\mathsf{st}(i)$, we proceed this time down the tree, with each inner node receiving from its parent a partial opening down to its index, extending it to an opening for each of its children's indices, and sending each extended opening to the corresponding child. Each inner node is able to extend the opening because it is the one that hashed all of its children's values together. Eventually, each machine $i \in [n]$ receives from its parent a complete opening down to index i. Finally, we construct a SNARG proof for the alleged RAM computation $(\mathsf{st}(1), \ldots, \mathsf{st}(n)) \to (\mathsf{st}'(1), \ldots, \mathsf{st}'(n))$. To do so, we first observe that this transition is defined by n separate computations, and for each $i \in [n]$, we prove that when the network state assignment is $(\mathsf{st}(1), \ldots, \mathsf{st}(n))$, the next state of machine i is $\mathsf{st}'(i)$. Each such statement can be proved by a SNARG computed at machine i, as follows. Let j be a machine that sends message m_j^i to machine i during the round. Recall that after the previous stage, machine j

[3] We assume for simplicity that the messages each machine sends are stored as part of its state at the beginning of the round.

has an opening from the global hash to its state $\mathsf{st}(j)$. Machine j now computes an opening from $\mathsf{st}(j)$ to m_j^i and sends this opening to machine i.Each machine, upon receiving these openings, uses a RAM SNARG to prove that its transition $\mathsf{st}(i) \to \mathsf{st}(i+1)$ is legal.

The remaining challenge is to aggregate the individual SNARG proofs of the machines into one global succinct proof while maintaining soundness. To do this we use the virtual network again: we proceed up the tree, with each node sending its current proof up to its parent. Upon receiving the s proofs from its children, each inner node uses an UpBARG to obtain a proof of the *conjunction* of its childrens' statements. Finally, at the root of the tree, we obtain our one succinct proof for the entire transition $(\mathsf{st}(1), \ldots, \mathsf{st}(n)) \to (\mathsf{st}^+(1), \ldots, \mathsf{st}^+(n))$.

References

1 Eden Aldema Tshuva, Elette Boyle, Ran Cohen, Tal Moran, and Rotem Oshman. Locally verifiable distributed SNARGs. In *TCC*, pages 65–90. Springer, 2023. doi:10.1007/978-3-031-48615-9_3.

2 Eden Aldema Tshuva and Rotem Oshman. Fully Local Succinct Distributed Arguments. In *DISC*, pages 1:1–1:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DISC.2024.1.

3 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013. doi:10.1145/2488608.2488623.

4 Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential ddh. In *Crypto*, pages 635–668. Springer, 2023. doi:10.1007/978-3-031-38551-3_20.

5 Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *Annual International Cryptology Conference*, pages 394–423. Springer, 2021. doi:10.1007/978-3-030-84259-8_14.

6 Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *FOCS*, pages 68–79, 2021.

7 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. doi:10.1145/1327452.1327492.

8 Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *FOCS*, pages 1057–1068, 2022. doi:10.1109/FOCS54457.2022.00103.

9 Cody Freitag, Rafael Pass, and Naomi Sirkin. Parallelizable delegation from LWE. In *TCC*, pages 623–652, 2022. doi:10.1007/978-3-031-22365-5_22.

10 Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

11 Yael Kalai and Omer Paneth. Delegating ram computations. In *TCC*, pages 91–118. Springer, 2016.

12 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948. SIAM, 2010. doi:10.1137/1.9781611973075.76.

13 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *PODC*, pages 9–18, 2005. doi:10.1145/1073814.1073817.

14 Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *FOCS*, pages 1045–1056, 2022. doi:10.1109/FOCS54457.2022.00102.

15 Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, pages 1–18. Springer, 2008. doi:10.1007/978-3-540-78524-8_1.

16 Brent Waters and David J Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *Crypto*, pages 433–463. Springer, 2022. doi:10.1007/978-3-031-15979-4_15.

Brief Announcement: DAGs for the Masses

Michael Anoprenko
Institut Polytechnique de Paris, France

Andrei Tonkikh
Aptos Labs, New York, NY, USA

Alexander Spiegelman
Aptos Labs, New York, NY, USA

Petr Kuznetsov
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Abstract

A recent approach to building consensus protocols on top of Directed Acyclic Graphs (DAGs) shows much promise due to its simplicity and stable throughput. However, as each node in the DAG typically includes a *linear* number of references to the nodes in the previous round, prior DAG protocols only scale up to a certain point when the overhead of maintaining the graph becomes the bottleneck.

To enable large-scale deployments of DAG-based protocols, we propose a *sparse DAG* architecture, where each node includes only a *constant* number of references to random nodes in the previous round. We present a *sparse* version of Bullshark – one of the most prominent DAG-based consensus protocols – and demonstrate its improved scalability.

Remarkably, unlike other protocols that use random sampling to reduce communication complexity, we manage to avoid sacrificing resilience: the protocol can tolerate up to $f < n/3$ Byzantine faults (where n is the number of participants), same as its less scalable deterministic counterpart. The proposed "sparse" methodology can be applied to any protocol that maintains disseminated system updates and causal relations between them in a graph-like structure. Our simulations show that the considerable reduction of transmitted metadata in sparse DAGs results in more efficient network utilization and better scalability.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Consensus, Atomic Broadcast, Byzantine Fault Tolerance, DAGs, Scalability, Sampling

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.45

Related Version *Full Version*: `https://arxiv.org/abs/2506.13998` [2]

Funding *Petr Kuznetsov*: Supported by CHIST-ERA grant CHIST-ERA-22-SPiDDS-05 (REDONDA project) and Agence Nationale de la Recherche (ANR-23-CHR4-0009).

1 Introduction

Byzantine consensus is one of the central tasks in distributed computing and blockchains. It combines disseminating data blocks among the users and making sure that the blocks are placed by all users in the same order. A popular trend of DAG-based consensus ([14, 16, 15, 3, 4], to name a few) is to separate data dissemination from the ordering logic. In these protocols, all users disseminate blocks, and each block references a number of *parents* – previously disseminated blocks – thus forming a directed acyclic graph (DAG). The key observation is that by locally interpreting snapshots of this ever-growing graph, the users can infer a consistent ordering of the blocks. Compared to more traditional, leader-based solutions such as PBFT [8], this approach is simpler, exhibits a more stable throughput and scales to a larger number of participants.


© Michael Anoprenko, Andrei Tonkikh, Alexander Spiegelman, and Petr Kuznetsov;
licensed under Creative Commons License CC-BY 4.0



Table 1 Communication complexity of Bullshark and Sparse Bullshark for different cryptographic setups and the estimated amount of egress traffic per-user per-round in the case when $n = 2\,000$ and $\lambda = 128$.

	Bullshark		Sparse Bullshark	
	complexity	ex. egress traffic	complexity	ex. egress traffic
Threshold signatures	$\Theta(n^2\lambda)$	171 MB	$\Theta(n^2 + n\lambda^2)$	17 MB
Multi-signatures	$\Theta(n^3)$	837 MB	$\Theta(n^2\lambda)$	81 MB
Regular signatures	$\Theta(n^3\lambda)$	171 GB	$\Theta(n^2\lambda^2)$	16 GB

The original DAG-based BAB protocols are, however, very heavy on the communication channels, which inherently limits their scalability. In this paper, we take a *probabilistic* approach to build a slimmer DAG structure, where each block carries *quasi-constant* amount of data about its predecessors, which results in a much less bandwidth-demanding solution. We show that the reduced complexity comes with the cost of a very small probability of consistency violation and a slight latency increase.

DAG consensus

DAG-based consensus protocols proceed in (asynchronous) rounds. In each round, each *user* (validator) involved in maintaining the DAG creates a node (typically containing a block of transactions submitted by the blockchain users) and disseminates it using reliable broadcast [6, 7]. The node contains references to $n - f$ nodes from the previous round, where n is the total number of validators and f is the maximum number of faulty validators (typically, $f = \lfloor \frac{n-1}{3} \rfloor$). In every round, a dedicated node is chosen as an *anchor*, where the choice can be either made deterministically based on the round number or at random using a common coin. The goal of consensus is now to place the anchors in an ordered sequence. In creating the sequence, the DAG nodes referencing the anchor as their parent are interpreted as its *votes*. Once the sequence of anchors is built, all DAG nodes in an anchor's causal past are placed before it in some deterministic way.

The burden of metadata

The central claim for DAG-based protocols is that they are able to amortize the space taken by the graph edges (*metadata*) provided enough user transactions (*payload*): If each block includes $\Omega(n)$ user transactions, the metadata will take at most a constant fraction of the block space. However, while alluring in theory, this claim does not stand the test of reality where we cannot simply keep increasing the block size indefinitely due to limited resources (network, storage, and CPU) and a limited demand.

As each user needs to receive, process, and store each other user's block in each round, and each block includes $n - f$ references, per-user per-round communication, computational, and storage complexity is at least $\Omega(n^2)$ multiplied by the size of a single reference. In practice, in most protocols [14, 10, 16, 15, 3], each reference includes either $n - f$ signatures, a multi-signature with a bit-vector of size n, or a threshold signature [11, 5], which brings the per-user per-round complexity to $\Omega(n^3\lambda)$, $\Omega(n^3)$, or $\Omega(n^2\lambda)$ respectively, depending on the cryptographic primitives available, where λ is the security parameter. [1]

[1] Intuitively, one can think of λ as the number of "bits of security", i.e., as the logarithm of the number

This issue is a significant barrier not only for effective resource utilization, but also for the scalability of the system: modern production blockchain deployments operate with around 100 validators and poorly scale beyond that.

Our contribution

We introduce a simple but surprisingly efficient technique for reducing the number of edges in the DAG by randomly sampling a constant ($D = O(\lambda)$) number of parents for each DAG node from a quorum of nodes in the previous round. The communication complexity is thus brought down to $O(n^2\lambda^2)$ with plain signatures, $O(n^2\lambda)$ with multi-signatures, or $O(n^2 + n\lambda^2)$ with threshold signatures. We provide a detailed break-down of the communication complexity in Section 3.2. This significantly lowers the overhead of DAG-based protocols, enabling larger-scale deployments. The communication complexity comparison is presented in Table 1.

Notably, unlike prior solutions to scaling distributed systems with random sampling ([13, 9, 1], to name a few), our approach maintains the optimal resilience and tolerates up to $f < n/3$ Byzantine faults.

We present *Sparse Bullshark*, a variant of Bullshark [16] – a state-of-the-art DAG-based consensus protocol – and demonstrate with simulations that it achieves better scalability.

2 Protocol

2.1 System Model

We use the standard BFT system model. For simplicity, we assume $n = 3f + 1$ participants $\Pi = \{p_1, \ldots, p_n\}$ (or *validators*), up to f of which might be *Byzantine* faulty. We assume partial synchrony with a known upper bound Δ on the network delays after GST and a (polynomially) computationally bound adaptive adversary.

We implement the *Byzantine Atomic Broadcast* [14] (BAB) abstraction satisfying the standard properties: **Validity**, **Agreement**, **Integrity**, **Total order**.

2.2 Sparse Bullshark

Our protocol is based on the partially synchronous version of Bullshark [16, 17].

Pseudocodes of both Bullshark and Sparse Bullshark with highlighted modifications can be found in the full version of this paper[2].

2.2.1 Verifiable sampling

The key distinguishing feature of Sparse Bullshark is that a DAG node stores just a constant-size *sample* of nodes from the previous round instead of a full quorum. We denote the sample size by D. Intuitively, when a correct validator collects $n - f$ round-r nodes, it selects, uniformly at random, a subset of size D of them and creates a new node in the round $r + 1$ referencing the selected subset. The specific choice of D affects communication complexity, latency, and the security of the protocol. More specifically, the protocol is secure against computationally bounded adversaries as long as D is $\Omega(\lambda)$. We discuss the impact of the sample size on latency in detail in Section 3.1.

of operations required to break the cryptosystem. Most hash functions and digital signature schemes have output of size $O(\lambda)$. However, multi-signatures require additional $O(n)$ bits to store the set of nodes who contributed to the signature. In this paper, we assume $n \gg \lambda$.

Instead of taking a random sample, Byzantine validators might attempt to undermine safety and liveness of the protocol by selecting a set of D nodes from the previous round maliciously. To anticipate this behavior, we employ *verifiable sampling*: Each validator must provide a *proof* that it has indeed collected a quorum of $n - f$ nodes in a round and has fairly sampled D of them in a pseudo-random manner.

To enable such proofs, whenever validator p_i broadcasts a round-r block, it attaches $Sign_i(r)$, a signature on the current round number. To prove that a validator indeed has seen a quorum of $n - f$ nodes of round r, it provides an aggregated *multi-signature* on the round number r and a bitset encoding the set of validators it received the round-r blocks from. The validator then uses the multi-signature on the round number as the seed for a pseudo-random number generator, which in turn is used to choose D parents from the collected quorum. To verify the sample, any other validator simply needs to check the multi-signature and replay the pseudo-random sampling procedure.

This approach, while restricting the behavior of Byzantine nodes to some degree, still leaves open the possibility of *grinding attacks*: a malicious validator that receives more than $n - f$ round-r blocks may try many different subsets of size $n - f$ to find a more "favorable" sample. However, as long as $D = \Omega(\lambda)$, this attack is, intuitively, as efficient as any other naive attack on the cryptographic primitives, such as simply trying to recover a correct node's private key by a brute force search. We address it formally in [2].

2.2.2 Referencing the anchor

Since the references to the anchor in round r by the nodes in round $r + 1$ are used as votes for the direct commit rule, on top of selecting D random parents, if a correct node observers the anchor of the previous round, it always includes it into its edges. Notice that trying to enforce this rule on Byzantine users would be unhelpful as they can always simply pretend to not have received the anchor node.

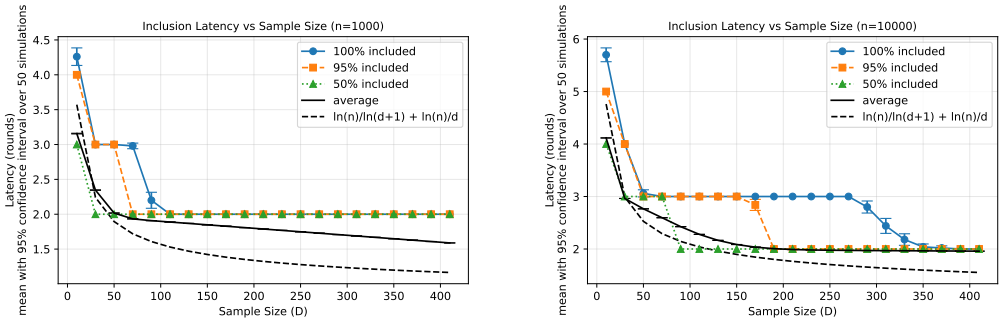
2.2.3 Direct commit rule

Finally, in Sparse Bullshark, we modify the direct commit rule, raising the threshold from $f + 1$ to $n - f = 2f + 1$. This does not affect the liveness or latency of the protocol in partial synchrony. Indeed, if the user producing the anchor node is correct and the network is in the stable period, all (at least $n - f$) correct users will reference the anchor node in round $r + 1$.

This modification is necessary in order to ensure that, with high probability, any committed anchor is reachable from any consequent anchor, and thus guarantee **Total order**.

3 Analysis

Correctness of Sparse Bullshark relies on the same key propery as Bullshark: if an anchor has been committed by an honest validator using the direct commit rule then there will be a path in DAG from any anchor of any future round to this anchor. This property is satisfied with high probability in Sparse Bullshark, implying the **Total order** property. We refer to the full version of the paper [2] for detailed correctness proofs.



(a) inclusion latency for $n = 1000$ nodes. **(b)** inclusion latency for $n = 10\,000$ nodes.

Figure 1 Inclusion latency (in DAG rounds) simulation, for varying sample size (D), assuming each node has D nodes from the previous round uniformly at random as its parents.

3.1 Latency

During periods of synchrony, all blocks created by correct validators in odd rounds will link to the anchor of the previous round because of the additional timer delay before creating a new vertex. Therefore, all anchor nodes by correct validators will be committed. The frequency of anchors being committed in Sparse Bullshark is the same as in the original Bullshark.

However, as the anchor node in Sparse Bullshark only references D nodes from the previous round instead of $n - f$, the protocol incurs additional *inclusion latency*: the delay between a vertex being broadcast and being included in some anchor's causal history. Assuming that each node has its parents selected at random from the previous round, the exact delay will follow the rumor-spreading pattern.

We provide a simulation of the inclusion delay (in terms of DAG rounds) in Figures 1a and 1b for 1000 and 10 000 users respectively. For example, selecting $D = 70$ for 1000 users or $D = 190$ for 10 000 users is enough to ensure that 95% of nodes will have inclusion latency of at most 2, i.e., will be referenced by some node referenced directly by the anchor. In contrast, Bullshark has inclusion latency of 1 for the $n - 1$ nodes referenced directly by the leader and 2 (assuming random message ordering, with high probability) for the rest of the nodes.

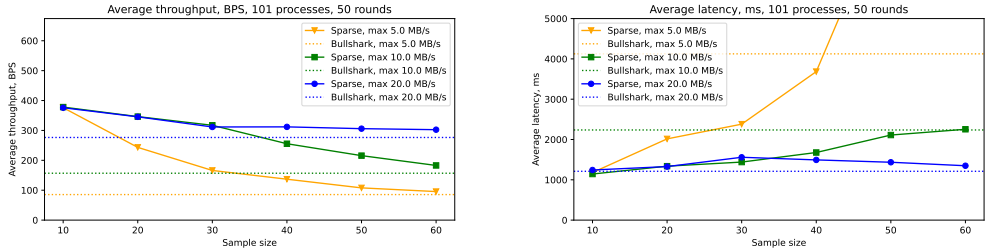
3.2 Communication complexity

We estimate the per-node metadata communication complexity. In Sparse Bullshark, each DAG node contains up to $D+2 = \Theta(\lambda)$ references to other nodes compared to $\Theta(n)$ references in Bullshark. Each reference is a certificate for the corresponding DAG node and its size is:

- $\Theta(\lambda)$ in case threshold signatures are available;
- $\Theta(n + \lambda) = \Theta(n)$ in case multi-signatures are available;
- $\Theta(n\lambda)$ otherwise.

We will denote the size of a certificate by C.

In Sparse Bullshark, the metadata of a node consists of $\Theta(\lambda)$ references and a bitmask of collected quorum (*sampleProof*), so metadata size for each node is $\Theta(n + \lambda C)$. In contrast, in Bullshark, the metadata consists of $\Theta(n)$ references, so the size of a node is $\Theta(nC)$.



(a) Throughput.

(b) Average commit latency.

Figure 2 Simulated statistics of Bullshark and Sparse Bullshark with various bandwidth limits and sample sizes.

Practical implementations rely on a combination of Signed Echo Consistent Broadcast [7] with randomized *pulling*[10, Section 4.1]. This approach allows us to surpass the theoretical bound on deterministic Reliable Broadcast communication complexity [12] and achieve linear expected communication complexity. With this approach, the expected communication complexity is $\Theta(n^2C)$ for Bullshark and $\Theta(n^2 + n\lambda C)$ for Sparse Bullshark. Refer to Table 1 for the exact complexity and example DAG node sizes depending on the choice of cryptographic tools.

3.3 Evaluation

We evaluated our Sparse Bullshark protocol using a simple simulator implemented in Python. We used emulated message delays with (50, 10)-normal distribution for 99% of messages and (500, 10)-normal distribution for a randomly selected 1% of the messages. We limited the network bandwidth of each simulated validator to emulate real life limitations of network bandwidth. Our goal was to evaluate the resources consumed by our protocol on the *metadata* exchange, compared to Bullshark [17].

Figure 2a shows the throughput of the protocol in blocks committed per second depending on the sample size for various per-node network bandwidth limits. We can observe that small sample sizes dramatically increase the throughput, allowing the protocol to be scalable when, under the same conditions, Bullshark saturates the network and does not scale.

Figure 2b shows the change in latency depending on the sample size for various per-node network bandwidth limits. Despite the theoretical result of increased latency in terms of message delays, we can see that in many cases under limited network bandwidth, commit latency actually decreases significantly due to sampling.

Evaluations confirm that while the sampling significantly decreases the amount of metadata transmitted through the network, which also leads to a decrease in latency when the bandwidth restriction is below a certain threshold.

Due to the limitations of the simulator implementation, we only tested a relatively small system size of 100 nodes with small sample sizes. However, the same trends will apply to larger systems, with proportionally larger sample sizes as well.

References

1 Ittai Abraham, Eli Chouatt, Ivan Damgård, Yossi Gilad, Gilad Stern, and Sophia Yakoubov. Asynchronous algorand: Reaching agreement with near linear communication and constant expected time. *Cryptology ePrint Archive*, 2025.

2 Michael Anoprenko, Andrei Tonkikh, Alexander Spiegelman, and Petr Kuznetsov. Dags for the masses, 2025. doi:10.48550/arXiv.2506.13998.

3 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast! *CoRR*, abs/2405.20488, 2024. doi:10.48550/arXiv.2405.20488.

4 Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the limits of latency with uncertified dags, 2024. arXiv:2310.14821.

5 Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2002.

6 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

7 Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

8 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002. doi:10.1145/571637.571640.

9 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a COINcidence: Sub-quadratic asynchronous byzantine agreement WHP. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.DISC.2020.25.

10 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

11 Yvo Desmedt. Threshold cryptosystems. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 1–14. Springer, 1992.

12 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985. doi:10.1145/2455.214112.

13 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017. doi:10.1145/3132747.3132757.

14 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

15 Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part I*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024. doi:10.1007/978-3-031-78676-1_6.

16 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

17 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: The partially synchronous version, 2022. doi:10.48550/arXiv.2209.05633.

Brief Announcement: Communication Patterns for Optimal Resilience

Hagit Attiya
Technion – Israel Institute of Technology, Haifa, Israel

Itay Flam
Technion – Israel Institute of Technology, Haifa, Israel

Jennifer L. Welch
Texas A&M University, College Station, TX, USA

Abstract

In response to the impossibility of solving the consensus problem in asynchronous systems subject to failures, various relaxations of the consensus problem have been proposed, including approximate agreement, crusader agreement, gather, and reliable broadcast. Some are interesting in their own right while others are useful building blocks for solving other problems. We focus on message-passing systems of n processes, up to f of which can experience malicious (Byzantine) failures. These problems all require that $n > 3f$ and they frequently have fairly simple and efficient algorithms when $n > 5f$. Challenges arise when considering resilience between $3f + 1$ and $5f$. For instance, nearly twenty years elapsed between the discovery of an approximate agreement algorithm for $n > 5f$ [10] and one for $n > 3f$ [1]. A stumbling block could be too much focus on looking for algorithms in a certain natural and intuitive form, which we call *canonical (asynchronous) rounds*. In such an algorithm, each process repeatedly sends a message containing its entire state tagged with a round number, then waits to receive $n - f$ messages with that same round number, does some local computation and proceeds to the next round number. The $n > 5f$ approximate agreement algorithm is in canonical round form *but the* $n > 3f$ *one is not.*

For algorithms in canonical round form, an obvious way of measuring time is the number of canonical rounds until the algorithm completes. However, this approach does not apply to other algorithms, such as those in which processes wait to receive a certain number of messages that have other properties besides simply having a certain round number. Attempts to rewrite these latter algorithms in canonical round form can result in drastically increased round complexity. This blow-up in the round complexity is inherent, as we show in this paper that *for a wide set of problems, there is no algorithm in canonical round form that has a finite upper bound on the number of rounds if* $n \leq 5f$. In contrast, the standard way of measuring time results in constant time complexity.

We first show the impossibility of a bounded number of canonical rounds for a generic problem that captures the key properties needed in the proof. The result then follows immediately for, most notably, crusader agreement and flavors of approximate agreement. We then show via reductions that the same result holds for reliable broadcast and gather, since there are crusader agreement algorithms that use reliable broadcast and gather with no round overhead.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases canonical rounds, reliable broadcast, gather, crusader agreement, approximate agreement, time complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.46

Related Version These results appear as part of `https://arxiv.org/abs/2510.04310`.

Funding *Hagit Attiya*: Supported by the Israel Science Foundation (22/1425 and 25/1849).
Itay Flam: Supported by the Israel Science Foundation (22/1425 and 25/1849).



© Hagit Attiya, Itay Flam, and Jennifer L. Welch;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

Modern geo-distributed systems are often modeled as a set of n *asynchronous* processes that communicate by sending and receiving messages; the possibility of arbitrary failures is captured by assuming that f of the processes might be *malicious* (*Byzantine*) and deviate from their algorithms or even collude. While traditional *consensus* cannot be solved in this model, even when $f = 1$ [14], relaxations of it are solvable, e.g., crusader agreement [9], approximate agreement [10], reliable broadcast [5], and gather [1, 7]. Solutions for these problems play a key role in Byzantine fault tolerance in asynchronous systems.

The *resilience* needed to solve a task in such a system is the relation between the total number of processes, n, and the maximal number of Byzantine processes, f. It is known that a resilience of $n > 3f$ is necessary and sufficient for solving many consensus-related problems. However, finding algorithms that have the optimal resilience can be challenging. For example, the seminal paper defining the approximate agreement problem [10] proved that $n > 3f$ is a lower bound on the resilience but the algorithms in that paper only have resilience $n > 5f$. Finding an algorithm with the optimal resilience, that is, an algorithm that only assumes $n > 3f$, has taken about twenty years [1].

We believe that a stumbling block to obtaining these algorithms is the inclination to look for algorithms that work in *canonical rounds*, a notion first mentioned by Fekete [11]. These are *full-information* algorithms, in which a process always sends all the information it has received so far, that proceed in *asynchronous rounds*. In each round r, a process p sends an r-message with the information it has received so far. Then, process p waits to receive r-messages from any $n - f$ distinct processes, changes state and proceeds to round $r + 1$. After some number of rounds S, process p decides on a value that is some function of the information it has received, and stops sending or receiving messages.

Assuming this communication pattern seems innocuous; indeed, what else can a process do? Fekete even claims that *"this form of protocol is completely general"* [12]. However, several widely-used algorithms employ variations on this communication pattern, by waiting to receive a certain number of messages *that satisfy a certain property*, not just any set of $n - f$ messages. For instance, in Bracha's reliable broadcast algorithm [5], the sender sends an *initial* message containing its input to all processes. Each process waits to receive an *initial* message, or $(n + f)/2$ *echo* messages for a common value, or $f + 1$ *ready* messages for a common value, and then sends an *echo* message for the value. Each process then waits to receive $(n + f)/2$ *echo* messages for a common value or $f + 1$ *ready* messages for a common value, and then sends a *ready* message for the value. Once a process has received $2f + 1$ *ready* messages for the same value, it accepts that value.

Bracha's algorithm can be restructured to use canonical rounds. Since processes must move to the next round after receiving $n - f$ messages for the current round, null messages are sent as placeholders while waiting to receive enough messages of a required type. The resulting algorithm reveals an interesting property: *The number of canonical asynchronous rounds required is unbounded.* The reason is that, even with a correct sender, the sender's *initial* message can be delayed arbitrarily long, while canonical rounds consisting largely of *null* messages continue to advance. In contrast, the canonical round version of Bracha's algorithm, like the original, has *bounded* (in fact, *constant*) time complexity, measured as the maximum real time elapsed until termination assuming every message sent between correct processes is delivered within one time unit [3,4]. The reason is that while Byzantine processes are "rushing" the correct processes through many canonical rounds, messages between correct processes are pending, and until they arrive less than one time unit can elapse.

This undesirable behavior is not caused simply by a poor conversion into canonical rounds. On the contrary, we prove in this paper that this behavior is inherent for a wide collection of fundamental problems. First, we show that any algorithm for a generic problem among $n \leq 5f$ processes has *an execution with an unbounded number of canonical rounds*. We then show that crusader agreement, approximate agreement on the real numbers, and approximate agreement on graphs are all special cases of the generic problem, which implies the same impossibility result for these problems. In particular, this explains why the code for the $n > 3f$ approximate agreement in [1] cannot be structured in canonical rounds. We observe that the approximate agreement algorithm for $n > 5f$ in [10] and the crusader agreement algorithm for $n > 5f$ in [4] both employ a bounded number of asynchronous canonical rounds, showing that requiring n to be at most $5f$ is necessary for the impossibility result.

To derive the same impossibility result for reliable broadcast and gather, we rely on the fact that they can be used to solve crusader agreement with little to no overhead.

To summarize, our main contribution is proving that canonical round algorithms for certain problems require an unbounded number of rounds, for Byzantine fault-tolerance with resilience $n \leq 5f$. The impossibility result holds for several important building blocks, including crusader agreement, reliable broadcast and gather, as well as for approximate agreement on the real line and on graphs. This points to the limitations of round-based asynchronous algorithms, both as a design principle and as a way to measure time, at least for resilience in the range $[3f + 1, 5f]$.

2 Canonical Round Algorithms have Unbounded Round Complexity

We assume the standard asynchronous model for n processes, up to f of which can be faulty, in which processes communicate via reliable point-to-point messages (cf., e.g., [4]). We consider *malicious* (or *Byzantine*) failures, where a faulty process can change state arbitrarily and send messages with arbitrary content.

Informally, an *execution* is a sequence of alternating configurations (states of the processes and in-transit messages) and events (WakeUp's and message deliveries). If α and β are executions and X is a set of processes, we say the executions are *indistinguishable* to X, denoted $\alpha \overset{X}{\sim} \beta$, if, for each process p in X, p has the same initial state and experiences the same sequence of events in α as in β.

An algorithm for n processes, f of which may be faulty, is in *canonical round* format and decides in S rounds if it operates as follows. Every message sent is labeled with the current round number of the sender, starting with 1. When a process wakes up, it sends a message containing its initial state to all the processes. Once a process receives $n - f$ messages for the current round, it increments its round counter and sends a message containing its initial state and the sequence of messages it has received to all the processes. Once a process reaches round S, it decides based on all the messages it has received.

Our result is proved for a **nontrivial convergence** problem in which there are at least two possible input values x_0 and x_1 and at least two decision values d_0 and d_1, such that if all correct processes have input x_i, then all correct processes must decide d_i, for $i = 0, 1$ (*validity*), and if a correct process decides d_0 in an execution, then no correct process can decide d_1 in the same execution (*agreement*).

▶ **Theorem 1.** *For any canonical round algorithm that solves a nontrivial convergence problem with $n \leq 5f$ and for any integer $K \in \mathbb{N}$, there exists an execution and a correct process that does not decide by round K.*

Proof. Assume towards contradiction that there exists a nontrivial convergence canonical round algorithm with $n \leq 5f$ and some $K \in \mathbb{N}$ such that in every execution, all correct processes decide by the end of round K.

For simplicity, we assume $n = 5f$, and divide the processes into five disjoint sets of f processes each: A, B, C, D, E. We consider the following initial configurations:

- Denote by C_0 the initial configuration such that processes in groups B, C, D, E are correct and processes in group A are Byzantine. All correct processes begin with input 0.
- Denote by C_1 the initial configuration such that processes in groups A, B, D, E are correct and processes in group C are Byzantine. All correct processes begin with input 1.
- Denote by C_2 the initial configuration such that processes in groups A, B, C, D are correct and processes in group E are Byzantine. Processes in groups B, C begin with input 0, and processes in groups A, D begin with input 1.

We construct three executions $\alpha_0, \alpha_1, \alpha_2$ starting at the initial configurations C_0, C_1, C_2 respectively, such that $\alpha_1 \overset{A,D}{\sim} \alpha_2 \overset{B,C}{\sim} \alpha_0$. Each execution is constructed as follows:

- α_0: The execution begins with WakeUp events for all processes in A, B, C, E; call this part of the execution α_0^0. Next appear $(n-f)^2$ receive events in which each of the $n-f$ processes in A, B, C, E receives the $n-f$ round 1 messages sent by the processes in A, B, C, E. Since $|A \cup B \cup C \cup E| = 4f = n - f$, the processes complete round 1 and send their round 2 messages. Call this part of the execution α_0^1. Similarly, define α_0^2 through α_0^K, so that processes receive round r messages and send round $r+1$ messages in α_0^r with the caveat that in α_0^K, processes decide instead of sending round $K+1$ messages. The processes in B, C, E, which are correct, send messages whose content is determined by the algorithm; the contents of the messages sent by the processes in A, which are Byzantine, are specified below. Note that processes in D take no steps in α_0 even though they are correct; consider them as starting late, after the other processes have completed K rounds.
- α_1: This execution and its partitioning into α_1^0 through α_1^K is defined analogously to α_0, but with processes in A, C, D, E exchanging messages, those in C being Byzantine, and those in B starting late.
- α_2: This execution and its partitioning into α_2^0 through α_2^K are similar to the previous executions but with some key differences. α_2^0 consists of WakeUp events for *all* the processes. α_2^1 consists of $(n-f)^2 + f$ receive events in which each of the $n-f$ correct processes receives a carefully selected set of $n-f$ round 1 messages and each faulty process takes a step in order to send a round 2 message. In particular, (correct) processes in A, D receive round 1 messages from processes in A, C, D, E, while (correct) processes in B, C receive round 1 messages from processes in A, B, C, E. Similarly, define α_2^2 through α_2^K. The contents of the messages sent by the (Byzantine) processes in E are defined below; unlike in α_0 and α_1, the round r messages sent to processes in A, D by a faulty process are not the same as those sent to processes in B, C by that process.

We now specify the messages sent by the faulty processes in each round r, $1 \leq r \leq K$, of the three executions. In α_0, a faulty process $p_i \in A$ sends the round r message sent by the corresponding correct process p_i in α_2. In α_1, a faulty process $p_i \in C$ sends the round r message sent by the corresponding correct process p_i in α_2. Finally, in α_2, a faulty process $p_i \in E$ sends to the correct processes in B, C the round r message sent by the corresponding correct process p_i in α_0, and to the correct processes in A, D the round r message sent by the corresponding correct process p_i in α_1.

Recall that $\alpha_i = \alpha_i^0 \dots \alpha_i^K$ for $i = 0, 1, 2$. Denote $\alpha_i^0 \alpha_i^1 \dots \alpha_i^r$ by $\alpha_i^{0:r}$ for $i = 0, 1, 2$ and $0 \leq r \leq K$. Claim 2, implying that α_0 and α_2 are indistinguishable to processes in B, C through round K, and Claim 3, implying that α_1 and α_2 are indistinguishable to processes in A, D through round K, can be proved by induction.

▷ Claim 2. For each $r, 0 \leq r \leq K$, (a) $\alpha_0^{0:r} \overset{B,C}{\sim} \alpha_2^{0:r}$ and (b) the same set of messages are in transit from A, B, C, E to B, C in the last configurations of $\alpha_0^{0:r}$ and $\alpha_2^{0:r}$.

▷ Claim 3. For each $r, 0 \leq r \leq K$, (a) $\alpha_1^{0:r} \overset{A,D}{\sim} \alpha_2^{0:r}$ and (b) the same set of messages are in transit from A, C, D, E to A, D in the last configurations of $\alpha_1^{0:r}$ and $\alpha_2^{0:r}$.

From the validity property of the nontrivial convergence problem, by the end of α_1, correct processes in groups A, D must decide 1. Since $\alpha_1 \overset{A,D}{\sim} \alpha_2$, the corresponding correct processes in these groups must decide 1 by the end of α_2. Similarly from validity, by the end of α_0 the correct processes in groups B, C must decide 0. Since $\alpha_0 \overset{B,C}{\sim} \alpha_2$, processes in groups B, C must decide 0 by the end of α_2 as well. This is in contradiction to the agreement property of the nontrivial convergence problem for execution α_2. ◀

3 Applications of Theorem 1

Crusader agreement [9] with input set V ensures that if all correct processes start with the same value $v \in V$, they must decide on this value, and otherwise, they may pick an *undecided* value, denoted $\perp$. If all correct processes start with $v \in \{0, 1\}$ they must decide v, and if a correct process decides $v \in \{0, 1\}$, the other correct processes decide either v or $\perp$. Therefore, crusader agreement is a nontrivial convergence problem with 0 and 1 being the two distinguished inputs and the two distinguished decisions, which implies:

▶ **Corollary 4.** *Any canonical round algorithm that solves crusader agreement with $n \leq 5f$ has an execution in which a correct process does not decide by round K, for any integer $K \in \mathbb{N}$.*

Crusader agreement is a special case of **connected consensus** [4], with parameter $R = 1$. It is easy to see that connected consensus for any $R \geq 1$ is a nontrivial convergence problem, and as a special case, so is **gradecast** [13].

Approximate agreement on the real numbers with parameter $\epsilon > 0$ [10] is defined as follows. Processes start with arbitrary real numbers and correct processes must decide on real numbers that are at most ϵ apart from each other. Decisions must also be *valid*, i.e., contained in the interval of the inputs of correct processes. To show approximate agreement is a nontrivial convergence problem, choose any two real numbers whose difference is greater than ϵ as the two distinguished inputs and two distinguished decisions. This implies:

▶ **Corollary 5.** *Consider a canonical round algorithm that solves ϵ-approximate agreement with $n \leq 5f$. If the input range includes v_0 and v_1 such that $|v_1 - v_0| > \epsilon$, then there is an execution in which some correct process does not decide by round K, for any integer $K \in \mathbb{N}$.*

A similar result can be shown for **approximate agreement on graphs** [8], provided that the graph has two vertices that are at distance 2 apart.

Reliable broadcast [5] and its weaker variant **consistent broadcast** [6] are defined with one designated *sender* process s. The sender has an input value v that it wants to broadcast. Processes may terminate by accepting a message from the sender. Consistent broadcast ensures that if the sender s is correct then eventually all correct processes accept

s's input (*validity*), and all correct processes that accept a value from sender s, accept the same value (*agreement*). Reliable broadcast further ensures that if some correct process accepts a value from s then eventually all correct processes accept a value from s (*relay*).

Consider the following algorithm for crusader agreement, assuming $n > 4f$, which uses n concurrent instantiations of consistent broadcast. Each process consistently broadcasts its input and then waits to accept values from $n - f$ consistent broadcast instances. If it accepts $n - 2f$ copies of some value v, then it decides v, otherwise it decides $\perp$. *Validity* for crusader agreement is immediate since when all correct processes start with v, they all accept at least $n - 2f$ copies of v and thus decide v. To argue *agreement*, assume for contradiction that one correct process accepts $n - 2f$ copies of v and another correct process accepts $n - 2f$ copies of $w \neq v$. By the agreement property of consistent broadcast, these messages are broadcast by different processes, and so $n \geq 2(n - 2f)$, which implies $n \leq 4f$, a contradiction.

Note that the algorithm simply waits for the termination of $n - f$ out of n concurrent invocations of consistent broadcast. Thus, a consistent broadcast algorithm that terminates within K canonical rounds would yield a crusader agreement algorithm that terminates within K canonical rounds, contradicting Corollary 4. This implies:

▶ **Corollary 6.** *Any canonical round algorithm for consistent or reliable broadcast, with $n \leq 5f$, has an execution in which a correct process does not terminate by round K, for any integer $K \geq 1$.*

Gather [1,2,7,13] is an extension of consistent broadcast in which *all processes* broadcast their value, and accept values from a large set of processes. Beyond properties inherited from consistent broadcast, most notably, that if two correct processes accept a value from another process, it is the same value, gather also ensures that there is a *common core* S^C of $n - f$ values that are accepted by all correct processes.

A very simple transformation shows that a gather algorithm can be used to solve crusader agreement, with no extra cost: Process p_i gathers the input values in a set S_i, and if some value v appears at least $|S_i| - f$ times in S_i, then it decides on v; otherwise, it decides on $\perp$. A counting argument shows that a value appearing $|S_i| - f$ times in a correct process p_i's variable S_i must appear at least $n - 2f$ times in the common core S^C. Since at most one value can appear $n - 2f$ times in S^C, all correct processes that decide on a non-$\perp$ value decide on the same value. Note that the transformation does not add any communication on top of the gather submodule. Thus, a gather algorithm that terminates within K canonical rounds would yield a crusader agreement algorithm that terminates within K canonical rounds, contradicting Corollary 4. This implies:

▶ **Corollary 7.** *Any canonical round algorithm for gather with $n \leq 5f$ has an execution in which some correct process does not terminate by round K, for any integer $K \geq 1$.*

4 Discussion

We have shown that for many important building blocks for Byzantine fault-tolerance with maximal resilience, canonical round algorithms require an unbounded number of rounds. Given that every round requires all-to-all communication, this also means an unbounded number of messages, although many of them are expected to be empty. For crash failures, there is an approximate agreement algorithm [10] that works in a logarithmic number of canonical rounds when $n > 2f$ (which is the optimal resilience). Thus the anomaly of requiring an unbounded number of canonical rounds when resilience is optimal does not (always) occur with crash failures.

References

1 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239. Springer, 2004. doi:10.1007/11516798_17.

2 Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 363–373. Association for Computing Machinery, 2021. doi:10.1145/3465084.3467914.

3 Hagit Attiya and Jennifer L. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004.

4 Hagit Attiya and Jennifer L. Welch. Multi-valued connected consensus: a new perspective on crusader agreement and adopt-commit. In *27th International Conference on Principles of Distributed Systems (OPODIS)*, pages 6:1–6:23, 2023. doi:10.4230/LIPICS.OPODIS.2023.6.

5 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

6 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 524–541. Springer, 2001. doi:10.1007/3-540-44647-8_31.

7 Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 42–51, 1993. doi:10.1145/167088.167105.

8 Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Convergence and covering on graphs for wait-free robots. *Journal of the Brazilian Computer Society*, 24:1–15, 2018. doi:10.1186/S13173-017-0065-8.

9 Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982. doi:10.1016/0196-6774(82)90004-9.

10 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986. doi:10.1145/5925.5931.

11 Alan D. Fekete. Asynchronous approximate agreement. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 64–76, 1987.

12 Alan D. Fekete. Asynchronous approximate agreement. *Information and Computation*, 115(1):95–124, 1994. doi:10.1006/INCO.1994.1094.

13 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997. doi:10.1137/S0097539790187084.

14 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

Brief Announcement: Highly Dynamic and Fully Distributed Data Structures

John Augustine
Indian Institute of Technology Madras, Chennai, India

Antonio Cruciani
Aalto University, Espoo, Finland

Iqra Altaf Gillani
National Institute of Technology Srinagar, India

Abstract

We study robust and efficient distributed algorithms for building and maintaining distributed data structures in dynamic Peer-to-Peer (P2P) networks. P2P networks are characterized by a high level of dynamicity with abrupt heavy node *churn* (nodes that join and leave the network continuously over time). We present a novel algorithmic framework to build and maintain, with high probability, a skip list for poly(n) rounds despite a *churn rate* of $\mathcal{O}(n/\log n)$, which is the number of nodes joining and/or leaving *per round*; n is the stable network size. We assume that the churn is controlled by an oblivious adversary that has complete knowledge and control of what nodes join and leave and at what time and has unlimited computational power, but is oblivious to the random choices made by the algorithm. Importantly, the maintenance overhead in any interval of time (measured in terms of the total number of messages exchanged and the number of edges formed/deleted) is (up to log factors) proportional to the churn rate. Furthermore, the algorithm is scalable in that the messages are small (i.e., at most polylog(n) bits) and every node sends and receives at most polylog(n) messages per round.

To the best of our knowledge, our work provides the first-known fully-distributed data structure and associated algorithms that provably work under highly dynamic settings (i.e., high churn rate that is near-linear in n). Furthermore, the nodes operate in a localized manner.

Our framework crucially relies on new distributed and parallel algorithms to merge two n-element skip lists and delete a large subset of items, both in $\mathcal{O}(\log n)$ rounds with high probability. These procedures may be of independent interest due to their elegance and potential applicability in other contexts in distributed data structures.

Finally, we believe that our framework can be generalized to other distributed and dynamic data structures including graphs, potentially leading to stable distributed computation despite heavy churn.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Networks → Peer-to-peer networks; Theory of computation → Data structures design and analysis

Keywords and phrases Peer-to-peer network, dynamic network, data structure, churn, distributed algorithm, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.47

Related Version *Full Version*: `https://arxiv.org/abs/2409.10235` [2]

Funding *Antonio Cruciani*: This work was supported in part by the Research Council of Finland, Grant 363558 and Partially supported by the Cryptography, Cybersecutiry, and Distributed Trust laboratory at IIT Madras (Indian Institute of Technology Madras) while visiting the institute.

1 Introduction

Peer-to-peer (P2P) networks underpin many modern distributed systems for resource sharing, storage, messaging, and content distribution (e.g., Gnutella, Skype, BitTorrent, Signal). They are highly dynamic: nodes join and leave continuously (*churn*), and links change arbitrarily,


© John Augustine, Antonio Cruciani, and Iqra Altaf Gillani;
licensed under Creative Commons License CC-BY 4.0



leading to frequent topological disruptions. Measurements of real-world P2P networks (see for example [10]) show that up to 50% of peers may be replaced within an hour, yet network size often remains relatively stable.

P2P systems support diverse tasks such as data storage and retrieval [16, 9], collaborative filtering [7], spam and malware detection [8, 18], and privacy-preserving data management [11]. Efficient distributed data structures such as distributed hash tables (e.g., CAN, Chord, Pastry, Tapestry) achieve good load balancing but offer little control over data placement and limited resilience to failures. Structured overlays like Skip Graphs [1], SkipNets [14], Rainbow Skip Graphs [12], and Skip$^+$ [15] allow ordered search and have been formally shown to be resilient to a limited number of faults (or equivalently small amounts of churn).

However, none of these data structures have theoretical guarantees of being able to work in a dynamic network with a very high adversarial churn rate, which can be as much as near-linear (in the network size) per round. This is a fundamental limitation for dynamic large-scale P2P systems, where it is essential to (i) preserve structural integrity, (ii) update quickly after joins and deletions, and (iii) answer queries reliably, all while keeping maintenance work proportional to the churn. Rebuilding from scratch is too costly; maintaining robustness under heavy churn without sacrificing simplicity or scalability remains an open challenge.

In this work, we address this gap by designing the first distributed skip list with rigorous guarantees under near-linear churn, ensuring fast updates, query correctness, and resource-competitive maintenance.

2 Preliminaries

Model: Dynamic Networks with Churn. We adopt the *dynamic network with churn* (DNC) model [4, 3]: a synchronous message-passing network of fixed size n, controlled by an *oblivious* adversary that may replace up to $\mathcal{O}(n/\log n)$ nodes per round after an initial $B = \beta \log n$-round *bootstrap phase* with no churn. Each node has a unique ID in $[0, poly(n)]$, stores one item, and can send/receive at most polylog(n)-bit messages over at most polylog(n) links per round. New nodes are connected to distinct existing nodes to avoid congestion. The adversary specifies a sequence $(V_t)_{t\geq 0}$ with $|V_t| = n$ for all t; joins and leaves are arbitrary subject to the churn bound. Nodes may create/delete edges; a bidirectional link (u, v) exists if both endpoints exchange invitation/acceptance messages, and is removed when either endpoint leaves or sends a delete message. For clarity, we focus here on the fixed-n, single-item case; in the full version, we generalize our algorithms to handle multiple items per node and a network size that may vary over time [2].

Problem Statement. We aim to maintain an *approximate distributed data structure* storing all items in the network, resilient to high adversarial churn. Insertions and deletions complete in $O(\log n)$ rounds, and membership queries $q(x, r, s)$ (initiated at source s in round r) must be answered within $Q = \mathcal{O}(\log n)$ rounds. A query returns "Yes" if x is present for the entire interval $[r, r + Q]$, "No" if absent for the entire interval, and may return either answer otherwise. This relaxed consistency, or *eventual correctness*, allows scalability and liveness under churn.

We require a *dynamic* formulation of *resource-competitiveness* [6]: the total work (number of messages and edge addition/deletions) over any interval is within a polylog(n) factor of the churn in that interval (allowing an $\mathcal{O}(\log n)$ lookback to capture delayed effects).

We fist show how to maintain an approximate distributed skip list, and then we extend our techniques to skip graphs and, in the full version, to general pointer-based distributed data structures.

3 Our Contribution

We first present our main result: the design and analysis of a churn-resilient distributed skip list with provable guarantees under near-linear adversarial churn. We then show how our techniques extend to more complex structures, including skip graphs, and, more generally, any pointer-based distributed data structure; we refer to the extended version for full details [2].

Main Result. Our dynamic distributed skip list data structure is architected using multiple "virtual networks" (see Figure 1). Each peer node can participate in more than one network and in some cases more than one location within the same network. We use the following network structures.

The Spartan Network $\mathcal{S}$ is a wrapped butterfly network that contains all the current nodes. This network can handle heavy churn of up to $\mathcal{O}(n/\log n)$ nodes joining and leaving in every round [5]. However, this network is not capable of handling search queries.

Live Network $\mathcal{L}$ is the skip list network on which all queries are executed. Some of the nodes in this network may have left. We require such nodes to be temporarily represented by their replacement nodes (from their respective neighbors in $\mathcal{S}$).

Buffer Network $\mathcal{B}$ is a skip list network on which we maintain all new nodes that joined recently.

Clean Network $\mathcal{C}$ is a skip list network that seeks to maintain an updated version of the data structure that includes the nodes in the system.

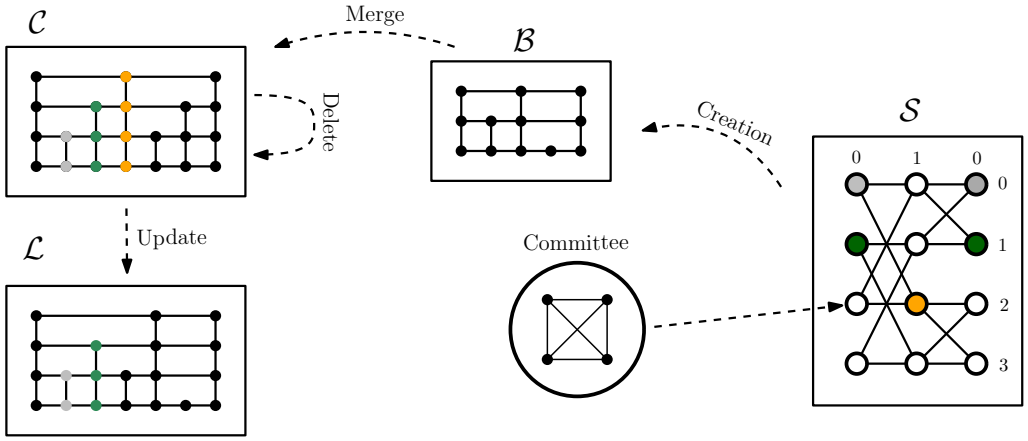


Figure 1 Schematic representation of the architecture and the Maintenance cycle described in Algorithm 1. Colored nodes in $\mathcal{L}$ and $\mathcal{C}$ are nodes that have been removed by the adversary and that are being covered by some committee of nodes (of the same color) in $\mathcal{S}$.

Moreover, in all skip lists, when a node exits the system, it is operated by a selected group of nodes. In the course of our algorithm description, if a node u is required to perform some operation, but is no longer in the system, then its replacement node(s) will perform that operation on its behalf. Note further that some of the replacement nodes themselves may need to be replaced. Such replacement nodes will continue to represent u. The protocol assumes a short ($\Theta(\log n)$ round) initial "bootstrap" phase, where there is no churn[1] and it

[1] Without a bootstrap phase, it is easy to show that the adversary can partition the network into large pieces, with no chance of forming even a connected graph.

initializes the underlying network. More precisely, the bootstrap is divided in two sub-phases in which we (i) build the underlying churn resilient network in $\mathcal{O}(\log n)$ rounds and, (ii) we build the skip lists data structures $\mathcal{L}$ and $\mathcal{C}$ (initially $\mathcal{L} = \mathcal{C}$) in $\mathcal{O}(\log n)$ rounds; after this, the adversary is free to exercise its power to add or delete nodes up to the churn limit and the network will undergo a continuous maintenance process. The overall maintenance of the dynamic distributed data structure goes through cycles. Each cycle $c \geq 1$ in Algorithm 1 consists of four sequential phases. Without loss of generality, assume that initially $\mathcal{L}$ and $\mathcal{C}$ are the same. We use the notation $\mathcal{L}^{(c)}$ (resp. $\mathcal{S}^{(c)}, \mathcal{C}^{(c)}, \mathcal{B}^{(c)}$) to indicate the network $\mathcal{L}$ (resp. $\mathcal{S}, \mathcal{C}, \mathcal{B}$) during the cycle $c \in \mathbb{N}$, when it is clear from the context we omit the superscript to maintain a cleaner exposition.

Algorithm 1 Overview of the distributed skip list maintenance process.

1 Starting from a skip list $\mathcal{L}^{(1)} = \mathcal{C}^{(1)}$
2 **foreach *Cycle*** $c \geq 1$ **do**
3 **Phase 1 (Deletion):** All the replacement nodes in $\mathcal{C}^{(c)}$ are removed. Note that nodes that leave the system during the replacement process may remain in $\mathcal{C}^{(c)}$.
4 **Phase 2 (Buffer Creation):** All the new nodes that were churned into the system since Phase 2 of the previous cycle join together to form $\mathcal{B}^{(c)}$.
5 **Phase 3 (Merge):** The buffer $\mathcal{B}^{(c)}$ created in Phase 2 is merged into $\mathcal{C}^{(c)}$, i.e., $\mathcal{C}^{(c+1)} \leftarrow \mathcal{C}^{(c)} \cup \mathcal{B}^{(c)}$.
6 **Phase 4 (Update):** Update the live network $\mathcal{L}^{(c)}$ with the clean network $\mathcal{C}^{(c)}$, i.e., $\mathcal{L}^{(c+1)} \leftarrow \mathcal{C}^{(c+1)}$.
7 **end**

Each phase of Algorithm 1 requires $\mathcal{O}(\log n)$ rounds w.h.p. In particular, the deletion and merge phases implement two novel parallel and fully distributed protocols to delete a batch of $\Theta(n)$ and merge two skip lists of $\Theta(n)$ elements in $\mathcal{O}(\log n)$ rounds w.h.p, respectively. This improves over prior $\mathcal{O}(k \log n)$-round bounds to perform these operation on a batch of k elements in skip-lists. We believe that some ideas from our results could be used in the centralized batch parallel setting to quickly insert batches of new elements in skip lists-like data structures (see e.g.,[17]) and in the fully dynamic graph algorithms settings (see for example the survey [13]) to perform fast updates of fully dynamic data structures. Indeed, in principle (provided the right amount of parallelism), our deletion and merge algorithms could be implemented in a parallel (centralized) setting and used to speed-up all kinds of computations involving these specific data structures. Our major contribution can be summarized in the following theorem.

▶ **Theorem 1** (Main Theorem). *Given a dynamic set of peers initially connected in some suitable manner (e.g., as a single path) that is stable for an initial period of $\mathcal{O}(\log n)$ rounds (i.e., the so-called* bootstrap phase*) and subsequently experiencing heavy adversarial churn at a churn rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round, there exists a resilient skip list, a distributed data structure that can withstand heavy adversarial churn at a rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round. We provide the following algorithms to support this data structure.*

- *An $\mathcal{O}(\log n)$ round algorithm to construct the resilient skip list during the bootstrap phase,*
- *a fully distributed algorithm that maintains the resilient skip list as nodes are inserted into or deleted from the data structure with the guarantee that the data structure reflects the updates within $\mathcal{O}(\log n)$ rounds,*

- *and a membership query algorithm that any peer can invoke and can be answered with efficiency parameter $Q \in \mathcal{O}(\log n)$.*

All nodes send and receive at most $\mathcal{O}(polylog(n))$ messages per round, each comprising at most $\mathcal{O}(polylog(n))$ bits. Moreover, such a maintenance algorithm is dynamically resource competitive. The maintenance protocol ensures that the resilient skip list is maintained effectively for at least n^d rounds with high probability, where $d \geq 1$ can be an arbitrarily chosen constant.

The above architecture and results extend to skip graphs and, to settings with multiple items per node and variable network size.

▶ **Corollary 2.** *Given a network with n nodes in which each vertex v possesses $t = polylog(n)$ elements in the skip list. Then maintenance protocol requires $\mathcal{O}(\log n)$ rounds to build and maintain a resilient skip list that can withstand heavy adversarial churn at a churn rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round.*

▶ **Corollary 3.** *The maintenance cycle described in Algorithm 1 can be adapted to support skip graphs. In particular, the resulting data structure remains resilient to churn at a rate of up to $\mathcal{O}(n/\log n)$ per rounds, and all operations complete within $\mathcal{O}(\log n)$ rounds with high probability, using at most $\mathcal{O}(polylog(n))$ messages per node per round.*

A General Framework for Churn-Resilient Structures. On a broader picture, the above results can be unified under a single general framework for maintaining pointer-based data structures under adversarial churn. Given a data structure $\mathcal{D}$, we can "abstract away" the ideas in Algorithm 1 and obtain the following *abstract maintenance cycle* for a generic data structure $\mathcal{D}$:

1. **Delete Phase:** Identify and remove corrupted, outdated, or disconnected regions of $\mathcal{D}$. Deletion ensures that inconsistencies caused by churn do not propagate through the structure.
2. **Creation Phase:** Organize the set of newly arrived nodes into a provisional structure $\mathcal{B}$ (the buffer data structure). The goal is to prepare these nodes for integration, typically by arranging them into a sorted or partially structured form.
3. **Merge Phase:** Integrate the buffer network $\mathcal{B}$ into the main structure $\mathcal{D}$ using a distributed merge protocol. This phase reconstructs the structure while respecting existing invariants.
4. **Update Phase:** Perform any necessary local corrections, including pointer rebalancing, level adjustments, or redundancy restoration, to finalize the integration.

Provided that we have $\mathcal{O}(T)$-round distributed algorithms for each phase of the above abstract maintenance cycle, we can maintain $\mathcal{D}$ against an adversarial churn rate of $\mathcal{O}(n/T)$ per round.

▶ **Theorem 4.** *Let $\mathcal{D}$ be a distributed pointer-based data structure maintained using our four-phase cycle, and let T be the maximum number of rounds needed for any phase. Then $\mathcal{D}$ can tolerate an adversarial churn rate of up to $\mathcal{O}(n/T)$ nodes per round, while preserving global correctness for at least n^c rounds w.h.p., for any fixed constant $c > 0$.*

Our abstract maintenance cycle allows to define classes of churn-resilient data structures in the DNC model, where each class is characterized by the churn rate per round (thus the maintenance cycle runtime) that the data structure can tolerate while still supporting efficient update and query operations. Formally, let t be the maintenance cycle run time function. We say that a distributed data structure $\mathcal{D}$ is **t-maintainable** if it can be successfully maintained by a t-round maintenance cycle.

▶ **Observation 5.** *Under the above general framework, every t-maintainable distributed data structure tolerates an adversarial churn rate of $\mathcal{O}(n/t)$ per round.*

In this brief announcment, we showed that Skip-Lists and Skip-Graphs are both $\mathcal{O}(\log n)$-maintainable and that are robust against and oblivious adversarial churn rate of $\mathcal{O}(n/\log n)$ per round. We conjecture that our bounds are tight, in the sense that $\log n$ is also a lower bound for the maintenance cycles for these data structures. That is because, in order to beat the $\Omega(\log n)$ barrier we would need to solve distributed sorting faster than in $\log n$ rounds while maintaining dynamic resource competitiveness.

This classification also raises a number of intriguing open problems. For example, it remains unclear whether there exist distributed data structures in the DNC model that are $\log\log n$-maintainable, and more generally, how to characterize the precise boundaries between maintainability classes. A key challenge is to establish lower bounds on the maintenance cycle time for fundamental distributed data structures. Do entirely new data structures need to be designed to exploit faster maintenance cycles? We believe that our formulation of maintainability classes in the DNC model opens up a rich landscape for further exploration.

4 Concluding remarks and discussion

In this work we proposed the first churn resilient skip list that can tolerate a heavy adversarial churn rate of $\mathcal{O}(n/\log n)$ nodes per round. The data structure can be seen as a four networks architecture in which each network plays a specific role in making the skip list resilient to churns and keeping it continuously updated. Moreover, we provided efficient $\mathcal{O}(\log n)$ rounds resource competitive algorithms to (i) delete a batch of elements from a skip list (ii) create a new skip list and, (iii) merge together two skip lists. This last result is the first algorithm that can merge two skip lists (as well as a skip list and a batch of new nodes) in $\mathcal{O}(\log n)$ rounds w.h.p.. We point out that these algorithms can be easily adapted to work on skip graphs [1, 12, 15].

In a broader sense, our technique is general and can be seen as a framework to maintain any kind of distributed data structure despite heavy churn rate. The only requirement is to devise efficient *delete, buffer creation, merge*, and *update* algorithms for the designated data structure. Furthermore, this allows us to define complexity classes for the maintenance of distributed data structure in the Dynamic Networks with Churn Model. Indeed, in this work we showed that skip list and skip graphs belong to the class of data structure that can tolerate a churn rate of $\mathcal{O}(n/\log n)$ per round.

Finally, given the simplicity of our approach, we believe that our algorithms could be used as building blocks for other non-trivial distributed computations in dynamic networks.

References

1 James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, 2007. doi:10.1145/1290672.1290674.

2 John Augustine, Antonio Cruciani, and Iqra Altaf Gillani. Maintaining distributed data structures in dynamic peer-to-peer networks, 2024. doi:10.48550/arXiv.2409.10235.

3 John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.29.

4 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the Twenty-Third*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012. SIAM, 2012. doi:10.1137/1.9781611973099.47.

5 John Augustine and Sumathi Sivasubramaniam. Spartan: Sparse robust addressable networks. *J. Parallel Distributed Comput.*, 2021. doi:10.1016/J.JPDC.2020.12.013.

6 Michael A. Bender, Jeremy T. Fineman, Mahnush Movahedi, Jared Saia, Varsha Dani, Seth Gilbert, Seth Pettie, and Maxwell Young. Resource-competitive algorithms. *SIGACT News*, 2015. doi:10.1145/2818936.2818949.

7 John F. Canny. Collaborative filtering with privacy. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*. IEEE Computer Society, 2002. doi:10.1109/SECPRI.2002.1004361.

8 Inc Cloudmark. Website of cloudmark. https://cloudmark.com/, last checked on 14-February-2024.

9 Peter Druschel and Antony I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany*. IEEE Computer Society, 2001. doi:10.1109/HOTOS.2001.990064.

10 Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas E. Anderson. Profiling a million user dht. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007*. ACM, 2007. doi:10.1145/1298306.1298325.

11 Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings*. USENIX Association, 2009. URL: http://www.usenix.org/events/sec09/tech/full_papers/geambasu.pdf.

12 Michael T. Goodrich, Michael J. Nelson, and Jonathan Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109601.

13 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 2022. doi:10.1145/3555806.

14 Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *4th USENIX Symposium on Internet Technologies and Systems, USITS'03, Seattle, Washington, USA, March 26-28, 2003*. USENIX, 2003. URL: http://www.usenix.org/events/usits03/tech/harvey.html.

15 Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip$^+$: A self-stabilizing skip graph. *J. ACM*, 2014. doi:10.1145/2629695.

16 Antony I. T. Rowstron and Peter Druschel. Storage management and caching in past, A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, Chateau Lake Louise, Banff, Alberta, Canada, October 21-24, 2001*. ACM, 2001. doi:10.1145/502034.502053.

17 Thomas Tseng, Laxman Dhulipala, and Guy E. Blelloch. Batch-parallel euler tour trees. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*. SIAM, 2019. doi:10.1137/1.9781611975499.8.

18 Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. Security applications of peer-to-peer networks. *Comput. Networks*, 2004. doi:10.1016/J.COMNET.2004.01.002.

Brief Announcement: Distributed Download from an External Data Source in Asynchronous Faulty Settings

John Augustine
Indian Institute of Technology Madras, Chennai, India

Soumyottam Chatterjee
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Valerie King
University of Victoria, Canada

Manish Kumar
Indian Institute of Technology Madras, Chennai, India

Shachar Meir
Weizmann Institute of Science, Rehovot, Israel

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Abstract

The distributed *Data Retrieval (DR)* model consists of k peers connected by a complete peer-to-peer communication network, and a trusted *external data source* that stores an array $\mathbf{X}$ of n bits ($n \gg k$). Up to βk of the peers might fail in any execution (for $\beta \in [0, 1)$). Peers can obtain the information either by inexpensive messages passed among themselves or through expensive queries to the source array $\mathbf{X}$. In the DR model, we focus on designing protocols that minimize the number of queries performed by any nonfaulty peer (a measure referred to as *query complexity*) while maximizing the resilience parameter β.

The Download problem requires each nonfaulty peer to correctly learn the entire array $\mathbf{X}$. Earlier work on this problem focused on *synchronous* communication networks and established several deterministic and randomized upper and lower bounds. Our work is the first to extend the study of distributed data retrieval to *asynchronous* communication networks. We address the Download problem under both the Byzantine and crash failure models. We present query-optimal deterministic solutions in an asynchronous model that can tolerate any fixed fraction $\beta < 1$ of crash faults. In the *Byzantine* failure model, it is known that deterministic protocols incur a query complexity of $\Omega(n)$ per peer, even under synchrony. We extend this lower bound to randomized protocols in the asynchronous model for $\beta \geq 1/2$, and further show that for $\beta < 1/2$, a randomized protocol exists with near-optimal query complexity. To the best of our knowledge, this is the first work to address the Download problem in asynchronous communication networks.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine Fault Tolerance, Blockchain Oracle, Data Retrieval Model, Distributed Download

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.48

Related Version *Previous Version*: `https://arxiv.org/abs/2412.19649` [3]

Funding *John Augustine*: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.

Soumyottam Chatterjee: Work done while at IIT Madras supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.

Manish Kumar: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.



© John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg;
licensed under Creative Commons License CC-BY 4.0



David Peleg: Venky Harinarayanan and Anand Rajaraman Visiting Chair Professor. The funds from this professorship enabled exchange visits between IIT Madras, India, and the Weizmann Institute of Science, Israel.

1 Introduction

1.1 Background and motivation

The Data Retrieval Model (DR) was first introduced in [2] to abstract the fundamental process of a group learning from a reliable external data source, where the data source is too large or it is too expensive to be learned individually (i.e., requires members of the group to collaborate), and some members of the group might crash during execution or act in other ways to deliberately sabotage the learning process. One key example of systems where this process takes place is *blockchain oracles* [6, 7]. We address this Oracle data delivery process in detail later and present a method for improving its performance using the DR model and the protocols designed in this work.

The DR model contains two entities: (i) a peer-to-peer network and (ii) an external data source in the form of an n bit array $\mathbf{X}$. There are k peers, up to β fraction of which may be faulty (and at least $\gamma = 1 - \beta$ fraction of which are nonfaulty). Each peer has access to the content of the array through queries. The general class of *retrieval problems* consists of problems $\mathsf{Retrieve}(f)$ requiring every peer to output $f(\mathbf{X})$ for some computable function f of the input array $\mathbf{X}$. In this work, we focus on the most fundamental retrieval problem, $\mathsf{Retrieve}(f_{id})$ where $f_{id}(\mathbf{X}) = \mathbf{X}$, referred to hereafter as the Download problem[1], where every peer needs to learn the entire input $\mathbf{X}$.

In [2] it is shown that *deterministic* Download in *synchronous* systems with Byzantine faults requires $\Omega(\beta n)$ queries, for every $\beta < 1$. This implies that in the presence of Byzantine faults, one cannot attain the ideal query complexity of $\frac{n}{\gamma k}$ without using randomization.

In this work, we consider Download protocols in the *asynchronous* setting for both the crash and Byzantine fault models. In the asynchronous Byzantine fault setting, we prove that, unlike the synchronous setting, where randomization can overcome the deterministic lower bound, $\Omega(n)$ queries per peer are required when $\beta \geq 1/2$, even for randomized protocols. We complement this lower bound with a protocol for the $\beta < 1/2$ regime that achieves a query complexity of $\tilde{O}\left(\frac{n}{(\gamma-\beta)k}\right)$, which, for a constant $\gamma - \beta$, is within log factors of the generic lower bound of $\Omega(n/\gamma k)$.

Turning our attention to the more benign setting of *crash* faults (i.e., where all peers are honest but some β fraction may stop functioning), the picture is brighter. For this model, it turns out that even in the asynchronous setting, one can get efficient *deterministic* Download protocols that achieve the optimal query complexity of $O\left(\frac{n}{\gamma k}\right)$, for any fraction $\beta < 1$ of crashes.

1.2 The Model

In the Data Retrieval (DR) model, the system consists of two components. The first is a collection of k peers, each equipped with a unique ID from the range $[1, k]$, connected by a complete communication network (or clique). The network provides *peer-to-peer message passing*, namely, every peer can send at time t a (possibly different) message of size at most ϕ bits to each other peer.

[1] It is fundamental since every retrieval problem $\mathsf{Retrieve}(f)$ can be solved by first performing download and then locally computing $f(\mathbf{X})$.

The second component of the DR model is an *external data source*. The *source* stores an n-bit input array $\mathbf{X} = \{b_1, \ldots, b_n\}$. It provides the peers with read-only access, allowing each peer to retrieve the data through queries of the form $\texttt{Query}(i)$, for $1 \leq i \leq n$. The answer returned by the source would then be b_i, the i^{th} element in the array. This type of communication is referred to as *source-to-peer* communication.

We consider asynchronous communication, where any communication (both among peer-to-peer and source-to-peer) can be delayed by any finite amount of time. For randomized protocols, we use the following notion of *cycles*.

Cycles. In the asynchronous model, there is no global notion of *rounds*, as each peer operates at a different pace. Nevertheless, to describe our protocols and analyze their performance, it is convenient to divide the *local* execution of each peer μ into (varying time) *cycles*. Each such local cycle consists of the following stages.

- Sending (0 or more) queries and getting answers.
- Sending (0 or more) messages.
- Waiting to receive messages.

We assume that local computation takes 0 time and can be performed at any point in a cycle. Moreover, when waiting for messages, after every message is received, the peer can adaptively decide whether to keep waiting for an additional message or continue to the next cycle. Note that the local cycle r of peer μ might coincide with a different local cycle r' of another peer μ'.

In the absence of global time units, it is convenient to break the time axis into "virtual blocks" by defining t^r, for integer $r \geq 1$, as the first time any peer started its local cycle r.

Every message is of size at most ϕ bits, where ϕ is a system parameter. Note that throughout the paper, we either set ϕ to a specific value, or leave it as a parameter, in which case increasing the message size parameter ϕ would result in faster protocols.

The adversary. Our analysis uses the notion of an *adversary*, representing the adverse conditions in which the system operates, including the asynchronous communication and the possibility of failures.

The adversary has two types of operation. First, it can fail up to βk peers, under the restriction that it can only fail a peer between its cycles (or before the first cycle), meaning that a peer can make random decisions in its current cycle without the adversary being able to react until the end of the cycle. Second, it can set the time $t^r_{\mu,\mu'}$ it takes a message sent by peer μ in its local cycle r to reach peer μ', under the restriction that it must set the time $t^r_{\mu,\mu'}$ for every pair of peers μ, μ', before time t^r. In other words, the adversary must set the latency of each message sent during a cycle r before any peer starts cycle r. The adversary can also decide when every peer starts its execution (i.e., we do not assume a simultaneous start). Note that in the case of deterministic protocols, the notion of cycles is irrelevant, and we consider a standard adversary that can fail a peer at any point of the execution and can delay messages for any finite amount of time.

The adversary $\mathcal{A}$ selects the input data and determines the failure pattern of the peers. In the *crash* failure model, the adversary's power is limited to crashing some of the peers in every execution of the protocol. Once a peer crashes, it stops its local execution of the protocol arbitrarily and permanently. This could happen *in the middle of operation*, e.g., after the peer has already sent some, but perhaps not all, of the messages it was instructed by the protocol to send out at a given point in time. In contrast, in the *Byzantine* failure model, a failed peer can deviate from the protocol in arbitrary ways. We assume that the adversary

can fail at most βk peers, for some given[2] $\beta \in [0, 1)$. We let $\gamma = 1 - \beta$, so there is (at least) a γ fraction of nonfaulty peers in every execution. Denote the set of faulty (respectively, nonfaulty) peers in the execution by $\mathcal{F}$. (resp., $\mathcal{H}$). We assume that the adversary knows the protocol and hence can simulate it (up to random coins).

We concentrate on the following complexity measures.

***Query* Complexity ($\mathcal{Q}$):** the maximum number of bits queried by a nonfaulty peer during the execution.

***Time* Complexity ($\mathcal{T}$):** the time it takes for the protocol to terminate.

***Message* Complexity ($\mathcal{M}$):** the total number of messages sent by nonfaulty peers during the execution.

We assume that queries to the source are the more expensive component in the system, so we focus mainly on optimizing the query complexity $\mathcal{Q}$. Measuring the *maximum* cost per peer (rather than the *total* cost) gives priority to a balanced load of queries over the nonfaulty peers.

Let us now formally define the Download problem. Consider a DR network with k peers, where at most βk can be faulty, and a source that stores a bit array $\mathbf{X} = [b_1, \ldots, b_n]$. Each peer is required to learn $\mathbf{X}$. Formally, each nonfaulty peer μ outputs a bit array res^μ, and it is required that, upon termination, $res^\mu[i] = b_i$ for every $i \in \{1, \cdots, n\}$ and $\mu \in \mathcal{H}$.

In the absence of failures, this problem can be solved by sharing the task of querying all n bits evenly among the k peers, yielding $\mathcal{Q} = \Theta(n/k)$. The message complexity is $\mathcal{M} = \tilde{O}(nk)$, assuming small messages of size $\tilde{O}(1)$, and the time complexity is $\mathcal{T} = \tilde{O}(n/k)$ since $\Omega(n/k)$ bits need to be sent along each communication link when the workload is shared.

1.3 Contributions

We present the Download problem in asynchronous communication networks, under both crash and Byzantine failures settings. In the crash-fault setting, our deterministic results are optimal w.r.t. to the resilience (for any $\beta < 1$) and query complexity. Notice that this optimality also holds for randomized algorithms. In the Byzantine failure setting, we provide deterministic and randomized lower bounds as well as upper bounds. The main results are the following (more details can be found in [3]).

(1) Deterministic Download **in Crash-Fault:** We present a deterministic protocol for solving Download problem in the asynchronous setting with at most $f < k$ crash faults ($\gamma = 1 - f/k$) with $\mathcal{Q} = \Theta(\frac{n}{\gamma k})$, $\mathcal{T} = O\left(\frac{n}{\phi} + \log_{k/f}(\phi)\right)$ and $\mathcal{M} = O(nk^2)$ where ϕ is the message size. Our result achieves the optimal query complexity for any fraction of crash fault, $\beta < 1$.

(2) Deterministic Lower Bound in Byzantine Fault: We show that for $\beta \geq 1/2$, every deterministic asynchronous Download protocol that is resilient to Byzantine faults requires $\mathcal{Q} = n$.

(3) Deterministic Download **in Byzantine Fault:** We show that for $\beta < 1/2$, there exists a deterministic asynchronous protocol that solves Download with $Q = O(\beta n)$, $\mathcal{T} = O\left(\frac{\beta n}{\phi}\right)$ and $\mathcal{M} = O(f \cdot n)$.

(4) Randomized Lower Bound in Byzantine Fault: We show that for any randomized asynchronous Download protocol where $\beta \geq 1/2$, there does not exist any execution in which every peer queries less than or equal $n/2$ bits.

[2] We do not assume β to be a fixed constant (unless mentioned otherwise).

(5) 2-cycle Randomized Download **in Byzantine Fault:** We present a 2-cycle asynchronous randomized protocol for Download with $\mathcal{Q} = O\left(\sqrt{\frac{n}{\gamma-\beta}} + \frac{n \log n}{(\gamma-\beta)k}\right)$ and $\mathcal{M} = O(k^2)$ where $\beta \leq 1/2$, and the message size is $\phi = O(\frac{n}{\gamma k})$.

(6) Randomized Download **in Byzantine Fault:** We present a $O\left(\log\left(\frac{\gamma k}{\ln n}\right)\right)$-cycle protocol that computes Download whp in the point-to-point model having expected query complexity $\mathcal{Q} = O\left(\frac{n \log n}{(\gamma-\beta)k}\right)$ and $\mathcal{M} = O\left(\log\left(\frac{\gamma k}{\ln n}\right) k^2\right)$ where $\beta \leq 1/2$, and the message size is $\phi = O(n)$.

To the best of our knowledge, this work is the first to study retrieval problems in the Data Retrieval (DR) model under asynchronous communication. The DR model has previously been explored in synchronous networks, most notably in [2] and the companion paper [4]. In particular, [2] introduced the Download problem, motivated by practical applications such as Distributed Oracle Networks (DONs), which form a crucial component of blockchain systems and employ protocols like OCR and DORA[6, 7].

2 Application: Efficient Blockchain Oracles

Blockchain systems [8] have seen a rise in popularity due to their ability to provide both transparency and strong cryptographic guarantees of agreement on the order of transactions, without the need for trusted third-party entities. More general computational abilities have also been well sought out for blockchains. Smart contracts [10] fulfill that need by providing users of the blockchain a way to run programs *on* the blockchain that ensures reliable and deterministic execution while providing transparency and immutability of both the code of the program and its state(s). Note that since the execution is required to be deterministic, i.e., every node must produce the same result, smart contracts are restricted to accessing *on-chain* data (that has been agreed upon), as *off-chain* data may introduce non-determinism to the execution.

Blockchain oracles [1, 5, 9] are components of blockchain systems that provide multiple services that support and extend the functionality of smart contracts (and other on-chain entities). The most important and fundamental service a blockchain oracle provides is bridging between the on-chain network and off-chain resources [6, 7], providing smart contracts access to external data without introducing non-determinism into their execution. We focus on this service and artificially consider it to be the sole responsibility of a blockchain oracle. In the remainder of this section, we explain in detail a possible application of the DR model and the Download problem for improved query efficiency within the context of *blockchain oracles*.

Blockchain oracles general structure. Blockchain oracles consist of an on-chain component and an off-chain component. The off-chain component encompasses the different data sources that store the required external information (e.g., stock prices, weather predictions) and the network of nodes in charge of retrieving that information and transmitting it to the on-chain component. The on-chain component can be thought to be (but is not necessarily) a smart contract that is responsible for verifying the validity of the report, making the information public on the blockchain it is hosted on, and using it for its execution.

Formally, the off-chain component consists of two parts: an asynchronous[3] *oracle network*, with peers (nodes) v_i, $i \in [1, k]$, capable of exchanging direct messages among themselves, and *data sources* DS_j, $j \in [1, m]$, each storing an array $\mathbf{X}_j$ of n variables in which the on-chain

[3] The network is sometimes assumed to be *partially synchronous* in blockchain oracles.

component is interested. Each peer can read the i-th cell from the j-th data source DS_j by invoking $\texttt{Query}(i, j)$. A fraction of up to $\beta_t \leq 1/3$ of the peers may be Byzantine, and a fraction of up to $\beta_d \leq 1/2$ of the m data sources may be Byzantine. Denote the on-chain component by SC.

The goal of blockchain oracles, as mentioned above, is to pull information from external sources and push a final value on-chain. There are a few difficulties that may arise when trying to develop such a system. First, it might be the case that different data sources report slightly different values (e.g., prices of a specific stock), even if all of them act honestly. Moreover, corrupted data sources might provide false and even inconsistent values (providing some nodes with value α and another with α'). The system needs to pick a final value in a way that (1) represents the range of honest values pulled from honest data sources and (2) malicious players (both data sources and peers) cannot force the system to pick a final value that does not represent the range of honest values.

The Oracle Data Delivery (ODD) problem. Denote by $\mathcal{H}_{ds}$ the set of honest data sources. Let $h_{\min}(i) = \min_{j \in \mathcal{H}}\{\mathbf{X}_j[i]\}$ and $h_{\max}(i) = \max_{j \in \mathcal{H}}\{\mathbf{X}_j[i]\}$. The *honest range* of i is the range $\sigma(i) = [h_{\min}(i), h_{\max}(i)]$. The *ODD problem* requires the on-chain SC to publish an array of values res to the target blockchain such that $res[i] \in \sigma(i)$, for every $i \in [n]$.

A blockchain oracle protocol can generally be split into three distinct steps: (1) collecting data, (2) reaching an agreement on the collected data, and (3) deriving and publishing a final value. Note that this abstraction is the minimum required abstraction to capture the operation of blockchain oracle protocols such as OCR [6] and DORA [7][4].

We now show how our Download protocols can be used to significantly reduce the cost of the *Oracle Data Collection (ODC)*, i.e, step (1) of blockchain oracles.

Improving ODC by blockchain oracles via Download**.** Current protocols perform the data collection step by the following ODC process:
For every node:

- Pick $2m\beta_d + 1$ data sources into a set ADS.
- Perform $o_{i,j} \leftarrow \texttt{Query}(i, j)$, for every $i \in [1, n]$ and $j \in ADS$.
- Calculate the median $o_i \leftarrow median(\{o_{i,j} \mid j \in ADS\})$ and proceed to step (2).

The results of [6, 7], cast in our abstract formulation, yield the following result.

▶ **Theorem 1.** **[6, 7]** *The ODC process guarantees that $o_i \in \sigma(i)$ for every $i \in [1, n]$ and has total query cost $O(mnk)$ and worst case individual query cost $\mathcal{Q} = O(mn)$.*

Instead, we propose utilizing the guarantees of Download protocols, namely, that for an honest data source DS_j, the output of each peer is exactly $\mathbf{X}_j$, to construct the following modification of the ODC steps.

- For every node, pick $2m\beta_d + 1$ data sources into a set ADS.
- For every data source $j \in ADS$, run a Download protocol (denote the result for cell i from data source j by $o_{i,j}$).
- Calculate the median $o_i \leftarrow median(\{o_{i,j} \mid j \in ADS\})$ and proceed to step (2).

[4] These protocols have many additional technical aspects, different structures, and different ways of handling steps (2) and (3). As our focus is on improving step (1), we may w.l.o.g. assume the abstract structure.

It is easy to verify that this modified construction yields the following.

▶ **Theorem 2.** *The* Download*-based ODC process guarantees $o_i \in \sigma(i)$ for every $i \in [1, n]$ and takes $\tilde{O}(mn)$ total queries and $\mathcal{Q} = \tilde{O}(mn/k)$ w.h.p.*

Note that the Download protocol presented in this paper assumes a binary input array, but this can be extended to numbers via a relatively simple extension. However, it is important to note that our solution relies on the following restrictive assumption. For two honest peers v, v', if both v and v' issue the query $\texttt{Query}(i, j)$, then they get the same result, for every $i \in [1, n]$ and honest data source j (i.e., the data does not change if queried at different times). Getting rid of this assumption and solving the problem efficiently for dynamic data is left as an open problem for future study.

References

1 John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A decentralized blockchain oracle. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1145–1152, 2018. doi:10.1109/Cybermatics_2018.2018.00207.

2 John Augustine, Jeffin Biju, Shachar Meir, David Peleg, Srikkanth Ramachandran, and Aishwarya Thiruvengadam. Byzantine Resilient Distributed Computing on External Data. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.3.

3 John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg. Distributed download from an external data source in faulty majority settings. *CoRR*, abs/2412.19649, 2024. doi:10.48550/arXiv.2412.19649.

4 John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg. Distributed Download from an External Data Source in Byzantine Majority Settings. In *DISC*, 2025. These proceedings.

5 Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tram‘er, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Technical report, Chainlink Labs, 2021.

6 Lorenz Breidenbach, Christian Cachin, Alex Coventry, Ari Juels, and Andrew Miller. Chainlink off-chain reporting protocol. Technical report, Chainlink Labs, 2021.

7 Prasanth Chakka, Saurabh Joshi, Aniket Kate, Joshua Tobkin, and David Yang. DORA: distributed oracle agreement with simple majority. *CoRR*, abs/2305.03903, 2023. doi:10.48550/arXiv.2305.03903.

8 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *whitepaper*, May 2009.

9 Supra Research. Supra's blockchain infrastructure stack. Whitepaper, Supra Labs, November 2024. URL: https://supra.com/documents/SupraTech-Whitepaper.pdf.

10 Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2, 1997. doi:10.5210/FM.V2I9.548.

Brief Announcement: Synchronization in Anonymous Networks Under Arbitrary Dynamics

Rida Bazzi
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Anya Chaturvedi
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Andréa W. Richa
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Peter Vargas
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Abstract

We present the δ-Synchronizer, which works in non-synchronous dynamic networks under minimal assumptions. Our model allows for arbitrary topological changes without any guarantee of eventual global or partial stabilization and assumes that nodes are anonymous. This deterministic synchronizer is the first that enables nodes to simulate a dynamic network synchronous algorithm for executions in a semi-synchronous dynamic environment under a weakly-fair node activation scheduler, despite the absence of a global clock, node ids, persistent connectivity or any assumptions about the edge dynamics (in both the synchronous and semi-synchronous environments). We make the following contributions: (1) we extend the definition of synchronizers to networks with arbitrary edge dynamics; (2) we present the first synchronizer from the semi-synchronous to the synchronous model in such networks; and (3) we present non-trivial applications of the proposed synchronizer to existing algorithms. We assume an extension of the PULL communication model by adding a single 1-bit multi-writer atomic register at each edge-port of a node. We show that this extension is needed and that synchronization in our setting is not possible without it. The δ-Synchronizer operates with memory overhead at the nodes that is asymptotically logarithmic on the runtime of the underlying synchronous algorithm being simulated – in particular, it is logarithmic for polynomial-time synchronous algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Synchronization, Anonymous Dynamic Networks, Arbitrary Dynamics

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.49

Funding *Anya Chaturvedi*: NSF-CCF-2312537 and 2106917; U.S. ARO (MURI W911NF-19-1-0233).
Andréa W. Richa: NSF-CCF-2312537 and 2106917; and U.S. ARO (MURI W911NF-19-1-0233).
Peter Vargas: NSF-CCF-2312537 and 2106917; and U.S. ARO (MURI W911NF-19-1-0233).

1 Introduction

Modern distributed systems, such as wireless sensor networks, mobile peer-to-peer systems, and biologically inspired swarm networks, often exhibit constantly changing and unpredictable communication dynamics that make achieving coordinated behavior between agents challenging, even for simple tasks. Achieving coordinated behavior in such a setting can be further compounded by asynchrony and agent anonymity (no identifiers). To simplify the design of distributed algorithms, researchers developed *synchronizers* that can transform algorithms designed under strong synchrony assumptions into algorithms that work correctly under weaker assumptions [2, 19, 15]. Previous work on synchronizers considers systems in which agents have unique identifiers and the network is either static or is dynamic for some


© Rida Bazzi, Anya Chaturvedi, Andréa W. Richa, and Peter Vargas;
licensed under Creative Commons License CC-BY 4.0



time but eventually stabilizes. In contrast, this paper considers a system of anonymous agents that communicate through a network with *arbitrary dynamics*, i.e., with *no restrictions on topological changes*, including *no assumptions on eventual (local or global) stabilization.*

In such networks, three different synchrony models are considered [13]. In the *synchronous* model, time is divided into stages and all nodes are active in every stage. In the *semi-synchronous model* (see, e.g., [14]), time is also divided into stages, but some nodes might not be active in a given stage. In both models, an active node executes one action – which involves communication with their neighbors and a bounded amount of computation – per stage, and topology changes occur only at the beginning of a stage. In other words, the *semi-synchronous* mode is one where time is synchronous but nodes are activated asynchronously. In the *asynchronous* model, there is no synchronization of time nor node activations, so actions can take an arbitrary bounded amount of time to execute, and topological changes and node activations can happen at arbitrary times.

In this paper, we introduce our *Arbitrary-Dynamics Synchronizer*, or *δ-Synchronizer* for short. The δ-Synchronizer is the first *deterministic* synchronizer that allows algorithms designed for a synchronous anonymous network with arbitrary adversarial dynamics to execute correctly in a semi-synchronous anonymous, arbitrary dynamics network under a *weakly-fair node activation scheduler*. Specifically, *our δ-Synchronizer transforms any algorithm $\mathcal{A}_{sync}$ designed for a* synchronous *dynamic network under arbitrary edge-dynamics given by a time-varying graph $\mathcal{G}_{sync}$ into an algorithm $\mathcal{A}_{semi}$ that correctly simulates $\mathcal{A}_{sync}$ under a* weakly-fair scheduler *in a* semi-synchronous *dynamic network under arbitrary edge-dynamics given by a time-varying graph $\mathcal{G}_{semi}$.*

Unlike other synchronizers that assume eventual stabilization and for which one can compare executions of the original algorithm and the transformed algorithm on the same stabilized network, in our setting, there is no guarantee that the network ever stabilizes and there is no guarantee that $\mathcal{G}_{sync}$ and $\mathcal{G}_{semi}$ are identical, which should not be surprising. This necessitates a *non-triviality* requirement on synchronizers for networks with arbitrary dynamics. The overall requirements for such synchronizers are captured by the following three general conditions (in bold), which we then indicate how they are satisfied by our δ-Synchronizer. We formalize our guarantees in Section 4.

- **(Correctness) The simulated synchronous execution is valid**: We show that for any $\mathcal{G}_{semi}$ and $i \geq 0$, there exists a $\mathcal{G}_{sync}$ such that the state of each node at the end of phase i (where phases are maintained by the synchronizer) of the semi-synchronous execution of $\mathcal{A}_{semi}$ under $\mathcal{G}_{semi}$ and a weakly-fair scheduler is equal to the state of each node at the end of i-th step of a synchronous execution of $\mathcal{A}_{sync}$ under $\mathcal{G}_{sync}$. (Theorem 1)
- **(Non-triviality) Every possible outcome of a synchronous execution can be simulated**: We show that for any $\mathcal{G}_{sync}$ and $i \geq 0$, there exists a $\mathcal{G}_{semi}$ such that the state of each node at the end of step i of the synchronous execution of $\mathcal{A}_{sync}$ under $\mathcal{G}_{sync}$, is equal to the state of each node at the end of phase i in a semi-synchronous execution of $\mathcal{A}_{semi}$ under $\mathcal{G}_{semi}$ and a weakly-fair scheduler. (Theorem 2)
- **(Finite termination) If the synchronous algorithm always terminates in finite time, so do the simulated executions**: We show that the synchronous execution of $\mathcal{A}_{sync}$ terminates in finite time for $\mathcal{G}_{sync}$ if and only if the semi-synchronous execution of $\mathcal{A}_{semi}$ terminates in finite time for $\mathcal{G}_{semi}$, which implies the condition. (Theorem 3)

While the *non-triviality* requirement rules out trivial solutions, e.g., in which $\mathcal{G}_{semi}$ always contains no edges regardless of the actual dynamics, our transformation actually satisfies a **strong non-triviality requirement**: Any edge (u, v) that persists long enough for both nodes u and v to be activated at least once during a phase $i \geq 0$ of the semi-synchronous execution *must* be part of $\mathcal{G}_{sync}$ during the i-th step of the synchronous execution. In particular, if $\mathcal{G}_{semi}$ is static, then the simulation guarantees that $\mathcal{G}_{sync} = \mathcal{G}_{semi}$.

Table 1 Comparison to other synchronizers.

Protocol	Year	Mapping	Network Dynamics	Anonymous
α, β, γ-Synchronizers [2]	1985	Sync to Async	static	No
Afek et al. [1]	1987	Sync to Async	dynamic with eventual-quiescence	No
Awerbuch and Sipser [5]	1988	Sync to Async	dynamic with t_π-stabilization	No
Awerbuch and Peleg [4]	1990	Sync to Async	static	No
Awerbuch et al. [3]	1992	Sync to Async	dynamic with eventual-quiescence	No
ζ-Synchronizer [20]	1994	Sync to Async	static	No
η_1, η_2, θ-Synchronizers [19]	1994	Sync to Async	static	No
σ-Synchronizer [17]	2020	Sync to Async	complete, static	No
Ghaffari and Trygub [15]	2023	Sync to Async	static	No
δ-Synchronizer (this paper)	2025	Sync to Semi-sync	**Arbitrary Dynamics**	**Yes**

We assume the PULL model of communication (see, e.g., [9]), with the addition of a *disconnection detector*, as in [12], and a *1-bit multi-writer atomic register at each edge-port*. The δ-Synchronizer operates with an asymptotic memory overhead that is logarithmic on the number of nodes provided that the underlying synchronous algorithm terminates in polynomial time [7]. We present applications and discuss future work in Section 5, and summarize the most relevant related work in Table 1.

2 Model

We consider an edge-dynamic network that we formalize using a time-varying graph [10]. A time-varying graph is represented as $\mathcal{G} = (V, E, T, \rho)$ where V is the set of nodes, E is a (static) set of undirected pairwise edges between nodes, with a lifetime $T \subseteq \mathbb{N}$. A *presence* function $\rho : E \times T \to \{0, 1\}$ indicates whether or not a given edge exists at a given time. We define a *snapshot* of $\mathcal{G}$ at time t as the undirected graph $G_t(V, E_t)$, where $E_t = \{e \in E : \rho(e, t) = 1\}$. At any time, we denote the *neighborhood* of a node $u \in V$ by $N(u)$. For $t \geq 0$, the t-th *stage* lasts from time t to the instant just before time $t + 1$; thus, the communication graph in stage t is G_t.

We assume that each node is equipped with a mechanism to detect disconnections on its ports [12, 16, 18]. When an edge connected to a node u is disconnected, the *disconnection detector* adds the corresponding port label to a temporary set D. The set D is reset to $\emptyset$ at the end of each activation of u. To ensure anonymity among nodes and their neighbors, each node u associates with its neighbors through port labels ℓ, noting that different nodes may connect to u through the same port over time, and that a node may connect at different ports of u over time (respecting that at most one node is connected to any port at any point in time). Each edge (u, v) is assigned specific ports at both nodes u and v. For convenience, we use $u.x$ to denote the value of *variable* x *at node* u.

An algorithm, including the synchronizer, is composed of multiple actions of the form $\langle label \rangle : \langle guard \rangle \to \langle operations \rangle$. An *action is enabled* if its guard (a boolean predicate) evaluates to true, and a *node* is said to be *enabled* if it has at least one enabled action. The scheduler controls the stages when an enabled node is activated and picks exactly one enabled action at the node to execute at the given stage. Our synchronizer algorithm will ensure that we only have one enabled action per node at any stage t.

We assume the PULL model of communication (see, e.g., [9]), where every time a node u is activated at a stage t, u can pull the state information of a neighboring node v. In [7], we show that the classic PULL (or PUSH) model is not powerful enough to support synchronization. Thus, we further equip the PULL model with a *1-bit multi-writer atomic register at each*

edge-port, and allow an activated node u at stage t to perform one atomic write onto each multi-writer register of any neighbor node v such that $(u, v) \in E_t$ during the stage. Each action follows the Read-Compute paradigm, where these two components are executed in this order in lockstep (and writes occur within the compute cycle).

We focus on *semi-synchronous* concurrency, where in each stage, any (possibly empty) subset of enabled nodes is activated concurrently, under arbitrary topological changes. To model this, we assume two adversaries, an *adaptive adversary* that controls the edge presence function at each stage (i.e, controls the edge dynamics), and the scheduler adversary – or simply the *scheduler* – that controls when nodes are activated. We assume a *weakly fair* scheduler that activates nodes such that any continuously enabled node is eventually activated (or equivalently, that every enabled node will be activated infinitely often).

Algorithm 1 δ-Synchronizer.

1: HANDSHAKE: $(\texttt{synch} = 0) \vee (\exists \ell \in P \setminus \tilde{D} \mid \ell.\texttt{block} = 0) \rightarrow$
2: **if** $\texttt{synch} = 0$ **then** ▷ If this is the initialization stage of synchronization
3: **for** $\ell \in N(u)$ **do** ▷ for a phase i, pull information from all neighbors
4: $X(\ell) \leftarrow \text{PULL}(\ell)$ ▷ and initialize $\tilde{D}$, the set of disconnected ports in
5: $\tilde{D} \leftarrow \phi$ ▷ the current phase to the empty set.
6: $P \leftarrow \{\ell \in N(u) \mid$ ▷ Valid neighbors of u to be considered in phase i's simulation are:
7: $[(X(\ell).\texttt{phase} < \texttt{phase})] \vee$ ▷ (1) neighbors that will catch up with phase i, or
8: $[(X(\ell).\texttt{phase} = \texttt{phase}) \wedge$ ▷ (2) neighbors simulating phase i that have either
9: $(X(\ell).\texttt{synch} = 0) \vee$ ▷ (2a) not initialized their valid neighbors, or
10: $(X(\ell).\texttt{port} \in X(\ell).P \setminus (X(\ell).\tilde{D} \cup X(\ell).D)))]\}$ ▷ (2b) also consider u as a valid ▷ persistent (i.e., not disconnected) neighbor in phase i.
11: $\texttt{synch} = 1$ ▷ Flag end of the initialization stage for phase i of the simulation.
12: **else** ▷ If u has finished initialization stage of the simulation of phase i,
13: **for** $\ell \in P \setminus (\tilde{D} \cup D)$ **do** ▷ For every persistent valid neighbor in the phase:
14: **if** $X(\ell).\texttt{phase} < \texttt{phase}$ **then** ▷ (a) Pull updated state from neighbors that are
15: $X(\ell) \leftarrow \text{PULL}(\ell)$ ▷ running behind and not yet simulating phase i.
16: **else** ▷ Otherwise, they must be simulating phase i, so
17: $X(\ell).\texttt{ack} \leftarrow \text{PULL}(\ell).\texttt{ack}$ ▷ (b) pull only neighbor's ack value for phase i.
18: $\tilde{D} \leftarrow \tilde{D} \cup D$ ▷ Add disconnections since last activation to the set $\tilde{D}$. ▷ After pulling state and ack information, attempt edge agreement.
19: **for** $\ell \in (P \setminus \tilde{D}) \mid [(X(\ell).\texttt{phase} = \texttt{phase}) \wedge (\ell.\texttt{block} = 0)]$ **do**
20: **if** $X(\ell).\texttt{ack} = 1$ **then** ▷ If a neighbor has already initiated edge agreement, block
21: BLOCK(ℓ) ▷ the edge by simultaneously setting the `block` flag on neighbor's port
22: $\ell.\texttt{block} = 1$ ▷ as well as on own port connected to edge to 1.
23: **else** ▷ Otherwise, indicate that state information for neighbor's
24: $\ell.\texttt{ack} = 1$ ▷ simulation of phase i has been pulled.

25: EXECUTESYNCH: $(\texttt{synch} = 1) \wedge (\ell.\texttt{block} = 1, \forall \ell \in P \setminus \tilde{D}) \rightarrow$
26: $F \leftarrow \{\ell \in P \mid (\ell.\texttt{block} = 1)\}$ ▷ F represents the final set of agreed neighbors of u in phase i
27: Run enabled action of (synchronous) algorithm $\mathcal{A}_{sync}$ with respect to $\{X(\ell) \mid \ell \in F\}$
28: $\texttt{phase} = \texttt{phase} + 1$
29: $\texttt{synch} = 0$ ▷ Clean up variables
30: **for** $\ell \in \{0, \ldots, \Delta\}$ **do** ▷ for new phase.
31: $\ell.\texttt{ack} = \ell.\texttt{block} = 0$

3 Algorithm

The δ-Synchronizer (Algorithm 1) consists of two actions, HANDSHAKE and EXECUTESYNCH, with exactly one of them being enabled for a node at any stage. Recall that the synchronizer works by showing the equivalence between a semi-synchronous execution of $\delta(\mathcal{A}_{sync})$ under an arbitrary dynamic network $\mathcal{G}_{semi} = \{G_0, G_1, \ldots\}$ and a synchronous execution of $\mathcal{A}_{sync}$ under a (potentially different) arbitrary dynamic network $\mathcal{G}_{sync} = \{H_0, H_1, \ldots\}$.

Each node u keeps a local counter, $u.\texttt{phase}$, of the phase number it is simulating and updates it when the simulation of the phase completes and an action of $\mathcal{A}_{sync}$ is executed (Lines 27-28). During the handshake for the simulation of phase i, a node u maintains a set of potential neighbors (identified by the ports to which they are connected) to be included as neighbors in the simulated execution, which is initially equal to the set $u.P$ of all valid neighbors at the start of the handshake, when $u.\texttt{synch} = 0$. The set of potential neighbors can "lose" elements in other stages of the handshake for phase i due to edge disconnections, but does not gain new elements. The crucial property to maintain is *edge consistency*: At the end of the simulation of phase i, the *final set* of neighbors of u, $u.F$, which is a subset of $u.P$, contains node v if and only if the set $v.F$ contains node u. The sets $u.F$, for all nodes $u \in V$, determine the set of (undirected) edges in the simulated graph H_i for phase i.

Initially, the set $u.P$ contains the following nodes:

1. (Running behind) Neighbors that have not yet started simulating phase i. These nodes can be included because node u will wait for them to catch up during the next stages in u's handshake for phase i (unless they disconnect from node u),

2a. (Concurrent nodes in initialization stages) Neighbors that have started simulating phase i but have not yet finished the initialization stage of the handshake. These nodes will also be able to see that node u has not yet finished its initialization stage of the simulation.

2b. (Concurrent nodes past initialization stages) Neighbors that completed the initialization of the simulation of phase i and are still considering node u as a neighbor in their handshake of phase i. These nodes must have started the handshake of phase i before node u but have not detected a disconnection in the edges linking them to node u.

To calculate the set $u.P$, a node pulls information from its neighbors and determines which nodes fall into one of the categories above (Lines 7–10). At the end of the first stage of the handshake, node u sets its synch flag to 1 to signal the end of its initialization for phase i.

To ensure *edge consistency*, each node u keeps track of all ports that have experienced a disconnection since the start of the simulation of phase i in its set $u.\tilde{D}$ (initially empty). All neighbors connected through ports in $u.\tilde{D}$ are removed from consideration to be included in the simulated graph H_i. Indeed, if there is a disconnection of an edge (u, v) in a given stage, then nodes u and v will each add the respective port connected to edge (u, v) to its own set $\tilde{D}$ in the first stage in which the node is active after the disconnection, and the edge (u, v) will not be included in the simulated graph H_i by either node. Since nodes are anonymous, any edge that later connects to the port of either u or v earlier connected to (u, v) – including, potentially, a reconnection of u and v – will be ignored in phase i.

A node u checks if the neighbors running behind in the simulation have caught up to phase i (Line 14) or, if the neighbor v has caught up, u only needs to update the respective ack value (since it already pulled the state information from v for phase i the last time it pulled from v in Line 15). Note that when the check is done in Line 14, if the neighbor is not behind, it must be simulating the same phase: The reason is that if the node was previously behind it could not have advanced beyond phase i without first executing the ack/block exchange and the first (and only) time that is done is when the two nodes are in phase i.

A node u completes the handshake of phase i when it determines that all edges linking it to a tentative neighbor in $u.P \setminus u.\tilde{D}$ are blocked (Line 25), implying that such an edge (u, v) will be considered to be in H_i by both u and v. The blocking of such an edge (u, v) can be initiated by u itself or by node v, but in either case it will result in the block flags on the respective ports at u and v being set to 1 during the current stage (while the edge is up). Whichever node blocks the edge-ports, say u, must have detected that the other node (v) set its ack to 1 (Lines 20-22), acknowledging that u and v are both simulating phase i, and that v has pulled u's phase i state information prior to the blocking (u has also pulled v's phase i state during the current stage). Note that there are blocked edges in phase i that disconnect later in the phase: Those edges will be in the respective sets F and thus also in H_i.

4 Our Results

Theorems 1–3, whose proofs appear in [7], ensure that the δ-Synchronizer satisfies the three conditions outlined in Section 1. The *$\mathcal{A}_{sync}$-state* of node u consists of the variables of u that directly pertain to the state variables of $\mathcal{A}_{sync}$.

▶ **Theorem 1** (Correctness). *For any semi-synchronous execution of $\delta(\mathcal{A}_{sync})$ under an arbitrary dynamic graph $\mathcal{G}_{semi}$ and a weakly-fair scheduler, there exists a dynamic graph $\mathcal{G}_{sync}$ such that the $\mathcal{A}_{sync}$-state of each node u at the end of each phase i of the execution of $\delta(\mathcal{A}_{sync})$ under $\mathcal{G}_{semi}$ is equal to the state of u at the end of step i in a synchronous execution of $\mathcal{A}_{sync}$ under $\mathcal{G}_{sync}$, for all $i \geq 0$.*

▶ **Theorem 2** (Non-triviality). *For any synchronous execution of $\mathcal{A}_{sync}$ under an arbitrary dynamic graph $\mathcal{G}_{sync}$, there exists a dynamic graph $\mathcal{G}_{semi}$ such that the state of each node u at the end of each step i of the synchronous execution of $\mathcal{A}_{sync}$ under $\mathcal{G}_{sync}$ is equal to the $\mathcal{A}_{sync}$-state of u at the end of phase i in the execution of $\delta(\mathcal{A}_{sync})$ under $\mathcal{G}_{semi}$, for all $i \geq 0$.*

▶ **Theorem 3** (Finite termination). *Our synchronizer ensures liveness: Any node progresses in finite time to phase i, for any finite $i \geq 0$. Thus, all synchronous executions of $\mathcal{A}_{sync}$ terminate in finite time if and only if all semi-synchronous executions of $\delta(\mathcal{A}_{sync})$ also do.*

5 Applications and Extensions

Designing algorithms for highly dynamic networks without any assumptions on edge dynamics or eventual stabilization is challenging, even in synchronous settings. On the other hand, scenarios with high and unpredictable network dynamics are getting increasingly more common in practice, and practitioners are looking at the benefits of time synchronization in order to better manage the dynamics (see, e.g., [11]). This work aims to bridge this divide.

In a classic application of our δ-synchronizer, we extend the applicability of the synchronous algorithm for maintaining a spanning forest that approximates the minimum possible number of spanning trees in arbitrary dynamic networks of [6] to semi-synchronous environments (after adapting the algorithm from the PUSH to PULL model). Another application is in the context of *minority dynamics* [8], a stateless protocol in which each node samples a random subset of neighbors and adopts the minority opinion observed. The impact of our synchronizer is not about enabling synchronous algorithms in semi-synchronous environments, but about providing an exponential speed-up in runtime when doing so, as we describe in [7].

In future work, we plan to investigate whether we can extend the δ-Synchronizer also to work in asynchronous environments, determine its overhead in terms of runtime and bits exchanged, and investigate whether it can work under certain classes of (non-stabilizing) network dynamics (e.g., edge recurrent, snapshot connected, etc.).

References

1 Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 358–370, 1987.

2 B. Awerbuch. Complexity of network synchronization. *Journal of ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

3 B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. E. Saks. Adapting to asynchronous dynamic networks. In *Proc. of the ACM Symp. on Theory of Computing (STOC)*, pages 557–570, 1992.

4 B. Awerbuch and D. Peleg. Network synchronization with polylogarithmic overhead. In *31st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 514–522, 1990.

5 B. Awerbuch and M. Sipser. Dynamic networks are as fast as static networks. In *29th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 206–220, 1988.

6 M. Barjon, A. Casteigts, S. Chaumette, C. Johnen, and Y. Neggaz. Maintaining a spanning forest in highly dynamic networks: The synchronous case. In *OPODIS*, pages 277–292, 2014.

7 R. Bazzi, A. Chaturvedi, A. W. Richa, and P. Vargas. Synchronization in anonymous networks under continuous dynamics. *CoRR*, 2025. doi:10.48550/arXiv.2506.08661.

8 L. Becchetti, A. Clementi, F. Pasquale, L. Trevisan, R. Vacus, and I. Ziccardi. The minority dynamics and the power of synchronicity. In *Proc. of the 2024 ACM-SIAM Symp. on Discrete Algorithms, (SODA)*, pages 4155–4176, 2024.

9 M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler. To push or to pull: On reducing communication and synchronization in graph computations. In *Proc. of ACM Intl. Symp. on High-Performance Parallel and Distributed Computing (HPDC)*, pages 93–104, 2017.

10 A. Casteigts. *A Journey through Dynamic Networks (with Excursions)*. Habilitation à diriger des recherches, U. of Bordeaux, 2018. URL: https://tel.archives-ouvertes.fr/tel-01883384.

11 J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8, 2013. doi:10.1145/2491245.

12 J. J. Daymude, A. W. Richa, and C. Scheideler. Local mutual exclusion for dynamic, anonymous, bounded memory message passing systems. In *Proc. of 1st Symp. on Algorithmic Foundations of Dynamic Networks (SAND)*, volume 221 of *LIPIcs*, pages 12:1–12:19, 2022. doi:10.4230/LIPICS.SAND.2022.12.

13 P. Flocchini, G. Prencipe, and N. Santoro, editors. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, volume 11340 of *LNCS*. Springer, Cham, 2019. doi:10.1007/978-3-030-11072-7.

14 P. Flocchini, N. Santoro, M. Yamashita, and Y. Yamauchi. A characterization of semi-synchrony for asynchronous robots with limited visibility, and its application to luminous synchronizer design. *CoRR*, abs/2006.03249, 2020. arXiv:2006.03249.

15 M. Ghaffari and A. Trygub. A near-optimal deterministic distributed synchronizer. In *Proc. of the 2023 ACM Symp. on Principles of Distributed Computing (PODC)*, pages 180–189, 2023.

16 V. Iyer, M. J. Pennell, J. T. West, and M. Beck. Dual termination serial data bus with pull-up current source. U.S. Patent No. 6593768 B1, 2003.

17 G. Pandurangan, D. Peleg, and M. Scquizzato. Message lower bounds via efficient network synchronization. *Theoretical Computer Science*, 810:82–95, 2020. doi:10.1016/J.TCS.2018.11.017.

18 R. Russell. Linux ethtool interface for controlling ethernet devices, 2001. URL: `https://man7.org/linux/man-pages/man8/ethtool.8.html`.

19 L. Shabtay and A. Segall. Low complexity network synchronization. In *8th Intl. Workshop on Distributed Algorithms (WDAG)*, volume 857 of *Springer LNCS*, pages 223–237, 1994. `doi:10.1007/BFB0020436`.

20 L. Shabtay and A. Segall. A synchronizer with low memory overhead (extended abstract). In *Proc. of the IEEE Intl. Conf. on Distributed Computing Systems*, pages 250–257, 1994.

Brief Announcement: Weaker Assumptions for Asymmetric Trust

Christian Cachin
University of Bern, Switzerland

Juan Villacis
University of Bern, Switzerland

Abstract

In protocols with asymmetric trust, each participant is free to make its own trust assumptions about others, captured by an asymmetric quorum system. This contrasts with ordinary, symmetric quorum systems and threshold models, where trust assumptions are uniformly shared among participants. Fundamental problems like reliable broadcast and consensus are unsolvable in the asymmetric model if quorum systems satisfy only the classical properties of consistency and availability. As a result, existing solutions introduce stronger assumptions to circumvent this limitation. We show that some requirements used by state-of-the-art approaches are overly restrictive, so much so that they effectively eliminate the benefits of asymmetric trust. To address this, we propose a new approach to characterize asymmetric problems and, building upon it, present an asymmetric asynchronous unauthenticated reliable broadcast algorithm that significantly weakens the assumptions needed to solve the problem. Our techniques are general and can be readily adapted to other core problems in the asymmetric trust setting.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Asymmetric Trust, Quorum Systems, Reliable Broadcast

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.50

Funding This work was supported by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 219403 (Emerging Consensus) and the Initiative for Cryptocurrencies and Contracts (IC3).

1 Introduction

Asymmetric trust models, such as those proposed by Damgård *et al.* [3], Alpos *et al.* [1], and Li, Chan, and Lesani [4] allow the development of distributed protocols in which each participant can operate under its own trust settings. These algorithms are built on top of asymmetric quorum systems, where each process independently defines its own quorums. The traditional consistency and availability properties must be satisfied by these systems, but as was shown by Li *et al.* [4], these are not enough to solve fundamental problems like reliable broadcast and consensus. Therefore, the models introduce additional assumptions. The protocols for reliable broadcast and consensus of Alpos *et al.* [1] require the existence of a *guild*, a set of processes that contains a quorum for each of its members. Li *et al.* [4] and Losa, Gafni, and Mazieres [5] identify similar conditions. These assumptions can be very strong and restrictive, and while they are sufficient to provide solutions to the aforementioned problems, we will show that they are not necessary. This leads to asymmetric algorithms that work under weaker assumptions, making such systems more flexible and usable.

We show that assumptions currently identified for algorithms such as reliable broadcast or consensus are so restrictive that it becomes unnecessary to use asymmetric trust models altogether. That is, given an asymmetric trust assumption that satisfies such requirements, we can build an equivalent symmetric trust assumption such that no process will be worse off by adopting it. This result is surprising, considering that Alpos *et al.* [1] show that asymmetric trust is more expressive than symmetric trust.


© Christian Cachin and Juan Villacis;
licensed under Creative Commons License CC-BY 4.0



To solve this, we introduce the notion of *depth* of a process and propose a new way to characterize asymmetric problems based on it. We use this concept to provide a more general definition of asymmetric asynchronous reliable broadcast and to propose a new algorithm for this problem that foregoes the *guild* requirement in favor of a weaker assumption on *depth* .

2 Asymmetric Trust Model

For a complete presentation of the asymmetric trust model, we refer the reader to the work of Alpos *et al.* [1]. In protocols with asymmetric trust, each participant is free to make its own individual trust assumptions about others, captured by an asymmetric quorum system. This contrasts with ordinary, symmetric and threshold quorum systems, where all participants share the same trust assumptions. Given a set of processes $\mathcal{P}$, an *asymmetric fail-prone system* $\mathbb{F} = [\mathcal{F}_1, \ldots, \mathcal{F}_n]$, where $\mathcal{F}_i$ represents the trust assumptions of process p_i, captures the heterogeneous model. Each $\mathcal{F}_i$ is a collection of subsets of $\mathcal{P}$ (the set of processes) such that some $F \in \mathcal{F}_i$ with $F \subseteq \mathcal{P}$ is called a fail-prone set for p_i and contains all processes that, according to p_i, may at most fail together in some execution [3]. We can in turn proceed to define the asymmetric Byzantine quorum systems. Here and from now on, the notation $\mathcal{A}^*$ for a system $\mathcal{A} \subseteq 2^{\mathcal{P}}$, denotes the collection of all subsets of the sets in $\mathcal{A}$, that is, $\mathcal{A}^* = \{A' | A' \subseteq A, A \in \mathcal{A}\}$.

▶ **Definition 1.** *An* asymmetric Byzantine quorum system $\mathbb{Q}$ *for* $\mathbb{F}$ *is an array of collections of sets* $\mathbb{Q} = [\mathcal{Q}_1, \cdots, \mathcal{Q}_n]$ *where* $\mathcal{Q}_i \subseteq 2^{\mathcal{P}}$ *for* $i \in [1, n]$. *The set* $\mathcal{Q}_i \subseteq 2^{\mathcal{P}}$ *is a symmetric quorum system of* p_i *and any set* $Q_i \in \mathcal{Q}_i$ *is called a quorum for* p_i. *The system* $\mathbb{Q}$ *must satisfy the following two properties.*

Consistency: *The intersection of two quorums for any two processes contains at least one process for which either process assumes that it is not faulty, i.e.,*

$$\forall i, j \in [1, n], \forall Q_i \in \mathcal{Q}_i, \forall Q_j \in \mathcal{Q}_j, \forall F_{ij} \in \mathcal{F}_i^* \cap \mathcal{F}_j^* : Q_i \cap Q_j \nsubseteq F_{ij}.$$

Availability: *For any process* p_i *and any set of processes that may fail together according to* p_i*, there exists a disjoint quorum for* p_i *in* $\mathcal{Q}_i$*, i.e.,*

$$\forall i \in [1, n], \forall F_i \in \mathcal{F}_i : \exists Q_i \in \mathcal{Q}_i : F_i \cap Q_i = \emptyset.$$

Given an asymmetric quorum system $\mathbb{Q}$ for $\mathbb{F}$, an asymmetric kernel system for $\mathbb{Q}$ is defined analogously as the array $\mathbb{K} = [\mathcal{K}_1, \ldots, \mathcal{K}_n]$ that consists of the kernel systems for all processes in $\mathcal{P}$. A set $K_i \in \mathcal{K}_i$ is called a *kernel* for p_i and for each K_i it holds that $\forall Q_i \in \mathcal{Q}_i,\ K_i \cap Q_i \neq \emptyset$, that is, a kernel intersects all quorums of a process.

The existence of a valid $\mathbb{Q}$ is conditioned on $\mathbb{F}$ satisfying the B^3 condition.

▶ **Definition 2** (B^3-condition)**.** *An asymmetric fail-prone system* $\mathbb{F}$ *satisfies the* B^3-condition*, abbreviated as* $B^3(\mathbb{F})$*, whenever it holds that*

$$\forall i, j \in [1, n], \forall F_i \in \mathcal{F}_i, \forall F_j \in \mathcal{F}_j, \forall F_{ij} \in \mathcal{F}_i{}^* \cap \mathcal{F}_j{}^* : \mathcal{P} \nsubseteq F_i \cup F_j \cup F_{ij}$$

Existing work shows that an asymmetric fail-prone system $\mathbb{F}$ satisfies $B^3(\mathbb{F})$ if and only if there exists an asymmetric quorum system for $\mathbb{F}$. If $B^3(\mathbb{F})$ holds, then the canonical quorum system, defined as the complement of the asymmetric fail-prone system, is a valid asymmetric quorum system.

During an execution the set of processes that fail is denoted by F. The members of F are unknown to the processes and can only be identified by an outside observer and the adversary. A process p_i correctly foresees F if $F \in \mathcal{F}_i^*$, that is, F is contained in one of its fail-prone sets. Based on this information, it is possible to classify processes in three categories.

Faulty: a faulty process, i.e., $p_i \in F$;
Naive: a correct process p_i, i.e. $p_i \notin F$, where $F \notin \mathcal{F}_i^*$; or
Wise: a correct process p_i, i.e. $p_i \notin F$, where $F \in \mathcal{F}_i^*$

Alpos *et al.* [1] show that naive processes might affect the safety and liveness guarantees of some protocols. In order to formalize this notion, they introduced the concept of a *guild*. This concept is central for many of the algorithms they propose (e.g., reliable broadcast, binary consensus), as properties can only be ensured for executions where a guild exists.

▶ **Definition 3.** *A guild is a set of wise processes that contains one quorum for each member. Formally, a guild $\mathcal{G}$ for $\mathbb{F}$ and $\mathbb{Q}$, for an execution with faulty processes F, is a set of processes that satisfies the following two properties.*

Wisdom: *$\mathcal{G}$ is a set of wise processes, that is,*

$$\forall p_i \in \mathcal{G} : F \in \mathcal{F}_i^*.$$

Closure: *$\mathcal{G}$ contains a quorum for each of its members, that is,*

$$\forall p_i \in \mathcal{G} : \exists Q_i \in \mathcal{Q}_i : Q_i \subseteq \mathcal{G}.$$

3 Guild Assumptions are not Asymmetric

Li *et al.* [4] show that given a Byzantine asymmetric quorum system, the classical properties of consistency and availability alone do not suffice to solve fundamental problems such as reliable broadcast or consensus. The heterogeneous views of the system make it necessary assume more structure. The corresponding algorithms of Alpos *et al.* [1] require the existence of a guild. In a similar manner, Li *et al.* [4] propose that the *strong availability* property should be satisfied to be able to solve such problems. This requirement closely resembles the notion of a guild; in fact, any execution that satisfies the strong availability property is guaranteed to contain at least one guild. We explore here the implications of requiring this kind of assumptions, focusing our attention on the model of Alpos *et al.* [1].

Guilds are a very strong assumption, essentially requiring all members to have common beliefs, which goes against the spirit of asymmetry. We show that given an asymmetric fail-prone system $\mathbb{F}$, in all executions where there is a guild, it is possible to construct an equally valid symmetric trust assumption $\mathcal{F}$ from $\mathbb{F}$. As a result, asymmetric trust reduces to symmetric trust.

In the crash-fault model, Senn and Cachin [6] already show that given an asymmetric fail-prone system $\mathbb{F}'$, it is possible to construct a symmetric fail-prone system $\mathcal{F}'$ from $\mathbb{F}'$ such that if all processes adopt $\mathcal{F}'$ as their trust assumption, no process will be worse off. This result implies that there is no need for asymmetric trust in that setting. We extend this by showing that a similar scenario occurs in the Byzantine model for all executions with a guild. This effectively invalidates the advantages of asymmetric trust for the algorithms proposed by Alpos *et al.* [1], which rely on the existence of a guild.

▶ **Theorem 4.** *Let $\mathbb{F}$ be an asymmetric fail-prone system. Given an execution with faulty processes F and at least one guild $\mathcal{G}$, it is possible to construct a symmetric fail-prone system $\mathcal{F}$ derived from $\mathbb{F}$ such that $F \in \mathcal{F}^*$.*

Theorem 4 shows that for executions with a guild, every process is at least as well off using the symmetric quorum system $\mathcal{F}$ instead of $\mathbb{F}$. For a wise process p_i the situation will remain the same regardless of the quorum system chosen. With the asymmetric system

$F \in \mathcal{F}_i^*$ will hold, while the symmetric alternative satisfies $F \in \mathcal{F}^*$. For naive processes the situation improves, as now the faulty processes will be considered in their trust assumption. Thus, there is no reason for a process not to adopt the derived symmetric fail-prone system $\mathcal{F}$. One limitation of this result is that it assumes knowledge of $\mathbb{F}$, the trust assumptions for all processes. Although Alpos *et al.* [1] also make this assumption, it is not required in other asymmetric trust models [5]. We also note that knowing the faulty processes F is not required to derive the symmetric fail-prone system presented in Theorem 4.

If a guild is needed to solve a problem in the asymmetric setting, there exists a way to use existing algorithms in the symmetric model and obtain the same guarantees that an asymmetric algorithm would provide. Therefore, if a problem can only be solved with a guild, the interest in using asymmetric algorithms decreases. This motivates to look for other models and algorithms to implement reliable broadcast, consensus, and other problems, where the asymmetric algorithms known so far require a guild.

4 Depth-Characterized Reliable Broadcast

The result of Section 3 motivates the search for new guildless algorithms for problems where the only known protocols require a guild. We explore this within the context of the Byzantine reliable broadcast problem. We present a new approach that significantly weakens the assumptions needed to implement the problem and removes the reliance on guilds.

In order to do this, we introduce the notion of the *depth* of a process. Intuitively, this captures the extent to which a process can depend on others during a multi-round protocol execution. We then show how it can be used to characterize asymmetric problems in a more fine-grained approach.

▶ **Definition 5** (Depth of a process). *For an execution, we recursively define the notion of a correct process having depth d as follows:*

- *Any correct process p_i has depth 0.*
- *Additionally, a correct process p_i has depth $d \geq 1$ if*

$$\exists Q \in \mathcal{Q}_i, \forall p_j \in Q : \; p_j \text{ is correct, has depth } s, \text{ and } s \geq d-1.$$

We will center our attention on the maximal depth of a process. Note that a process with maximal depth d also has depth d' for all $0 \leq d' \leq d$. Following the terminology of Alpos *et al.* [1], naive processes have maximal depth 0, wise processes have maximal depth at least 1, and processes in a guild have maximal depth ∞.

Definition 6 presents a way to characterize reliable broadcast based on the depth of processes. Its properties are specific to processes that have at least a certain maximal depth.

▶ **Definition 6** (Depth-characterized asymmetric asynchronous Byzantine reliable broadcast). *A protocol for* depth-characterized asymmetric (Byzantine) reliable broadcast *with sender p_s and depth d, shortened as RB[d], defined through the events dar-deliver and dar-broadcast, satisfies the following properties:*

- ***Validity:*** *If a correct process p_s* dar-broadcasts *a message m, then all processes with depth d eventually* dar-deliver *m.*
- ***Consistency:*** *If some process with depth d dar-delivers m and another process with depth d dar-delivers m', then $m = m'$*
- ***Integrity:*** *Every process with depth d dar-delivers m at most once. Moreover, if the sender p_s is correct and the receiver has depth d, m was previously dar-broadcast by p_s*
- ***Totality:*** *If a process with depth d* dar-delivers *some message, then all processes with depth d eventually* dar-deliver *a message.*

RB[∞] gives a solution for all processes with depth ∞, while the protocol by Alpos *et al.* only guarantees a solution for processes with depth ∞ that also belong to the maximal guild. Lemma 7 shows that if an algorithm solves *RB[s]* then it also solves *RB[s′]* for all $s' \geq s$. This simplifies the search for a solution by reducing it to finding an algorithm that works for the minimal value of s. Lemma 8 shows that there are no algorithms that can solve *RB[1]*, therefore we must search for a protocol that solves the problem for processes with depth ≥ 2.

▶ **Lemma 7.** *If an algorithm solves RB[s] then it also solves RB[s′] for all $s' > s$*

▶ **Lemma 8.** *There is no algorithm that solves RB[1]*

Algorithm 1 presents a solution for *RB[3]*. Every process in the algorithm waits to receive a quorum of READYAFTERECHO messages associated to the same message m and round r before delivering m. A process with depth 3 will only receive such a quorum if there exists a set of processes Q (which form a quorum for a wise process) that can attest that the sender indeed sent the value m. Since there will be an attesting quorum for every process that delivers, and since quorums for wise processes intersect in at least one correct process, we can deduce that all processes with depth 3 that deliver a value m will deliver the same value. In addition, we use a technique similar to Bracha's kernel amplification [2] to ensure that if a valid process delivers then all valid processes will deliver. The arrays in lines 4, 5, 6, and 7 are hashmaps, so even though they are depicted as having infinite size they are actually sparsely populated. The algorithm has a latency of 3 asynchronous rounds in the best case and 5 in the worst case scenario.

▶ **Lemma 9.** *Algorithm 1 solves RB[3]*

5 Conclusion

We have shown that if a problem requires a guild assumption to be solved it is of less interest to use asymmetric trust to solve it. This arises from Theorem 4, which shows that for these cases, asymmetric trust reduces to symmetric trust.

To address this we presented a more fine-grained approach to characterize asymmetric problems, using the concept of *depth*, and showed that reliable broadcast can be solved for all processes with depth at least 3. This weakens the requirements needed to solve the problem in an asymmetric setting compared to the solution proposed by Alpos *et al.* [1], which only solves reliable broadcast for a fraction of processes with depth ∞. It is an open question if there exists an algorithm that solves *RB[2]*. These techniques can also be applied to other problems, such as binary consensus and common coin, whose existing solutions in the asymmetric trust setting rely on guilds. This approach could enable the development of new algorithms that operate under significantly weaker assumptions.

Algorithm 1 Depth-based asymmetric reliable broadcast for processes with depth 3 with sender p_s ($RB[3]$) (process p_i).

State
  $sentecho \leftarrow$ FALSE: indicates whether p_i has sent ECHO
  $echos \leftarrow [\perp]^n$: collects the received ECHO messages from other processes
  $sentrae \leftarrow [\text{FALSE}]^\infty$: indicates whether p_i has sent READYAFTERECHO in round n, a hashmap
  $sentrar \leftarrow [\text{FALSE}]^\infty$: indicates whether p_i has sent READYAFTERREADY in round n, a hashmap
  $readysafterecho \leftarrow [\perp, \perp]^{\infty \times n}$: collects READYAFTERECHO messages from other processes, a hashmap
  $readysafterready \leftarrow [\perp, \perp]^{\infty \times n}$: collects READYAFTERREADY messages from other processes, a hashmap
  $delivered \leftarrow$ FALSE: indicates whether p_i has delivered a message

upon invocation *dar-broadcast*(m) **do**
  send message $[\text{SEND}, m]$ to all $p_j \in \mathcal{P}$

upon receiving a message $[\text{SEND}, m]$ from p_s **such that** $\neg sentecho$ **do**
  $sentecho \leftarrow$ TRUE
  send message $[\text{ECHO}, m]$ to all $p_j \in \mathcal{P}$

upon receiving a message $[\text{ECHO}, m]$ from p_j **do**
  if $echos[j] = \perp$ **then**
    $echos[j] \leftarrow m$

upon exists $m \neq \perp$ **such that** $\{p_j \in \mathcal{P} | echos[j] = m\} \in \mathcal{Q}_i$ **and** $\neg sentrae[1]$ **do** // a quorum for p_i
  $sentrae[1] \leftarrow$ TRUE
  send message $[\text{READYAFTERECHO}, 1, m]$ to all $p_j \in \mathcal{P}$

upon exists $m \neq \perp, r > 0$ **such that** $\{p_j \in \mathcal{P} | readysafterecho[r][j] = m\} \in \mathcal{K}_i$ **and** $\neg sentrar[r]$ **do**
  $sentrar[n] \leftarrow$ TRUE
  send message $[\text{READYAFTERREADY}, r, m]$ to all $p_j \in \mathcal{P}$

upon receiving a message $[\text{READYAFTERECHO}, r, m]$ from p_j **do**
  if $readysafterecho[r][j] = \perp$ **then**
    $readysafterecho[r][j] \leftarrow m$

upon receiving a message $[\text{READYAFTERREADY}, r, m]$ from p_j **do**
  if $readysafterready[r][j] = \perp$ **then**
    $readysafterready[r][j] \leftarrow m$

upon exists $m \neq \perp, r > 1$ **such that** $\{p_j \in \mathcal{P} | readyafterready[r][j] = m\} \in \mathcal{Q}_i$ **and** $\neg sentrae[r]$ **do**
  $sentrae[r+1] \leftarrow$ TRUE
  send message $[\text{READYAFTERECHO}, r+1, m]$ to all $p_j \in \mathcal{P}$

upon exists $m \neq \perp, r > 0$ **such that** $\{p_j \in \mathcal{P} | readysafterecho[r][j] = m\} \in \mathcal{Q}_i$ **and** $\neg delivered$ **do**
  $delivered \leftarrow$ TRUE
  output *dar-deliver(m)*

References

1 Orestis Alpos, Christian Cachin, Björn Tackmann, and Luca Zanolini. Asymmetric distributed trust. *Distributed Comput.*, 37(3):247–277, 2024. doi:10.1007/S00446-024-00469-1.

2 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985. doi:10.1145/4221.214134.

3 Ivan Damgård, Yvo Desmedt, Matthias Fitzi, and Jesper Buus Nielsen. Secure protocols with asymmetric trust. In *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2007. doi:10.1007/978-3-540-76900-2_22.

4 Xiao Li, Eric Chan, and Mohsen Lesani. Quorum subsumption for heterogeneous quorum systems. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 28:1–28:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.28.

5 Giuliano Losa, Eli Gafni, and David Mazières. Stellar consensus by instantiation. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.DISC.2019.27.

6 Michael Senn and Christian Cachin. Asymmetric failure assumptions for reliable distributed systems. In Davide Frey and Gowtham Kaki, editors, *Proceedings of the 12th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC 2025, World Trade Center, Rotterdam, The Netherlands, 30 March 2025- 3 April 2025*, pages 8–14. ACM, 2025. doi:10.1145/3721473.3722143.

Brief Announcement: Non-Uniform Content-Oblivious Leader Election on Oriented Asynchronous Rings

Jérémie Chalopin
Aix Marseille Univ, CNRS, LIS, Marseille, France

Yi-Jun Chang
National University of Singapore, Singapore

Lyuting Chen
National University of Singapore, Singapore

Giuseppe A. Di Luna
DIAG, Sapienza University of Rome, Italy

Haoran Zhou
National University of Singapore, Singapore

Abstract

In this paper, we study the leader election problem in oriented ring networks under content-oblivious asynchronous message-passing systems, where an adversary may arbitrarily corrupt message contents.

Frei et al. (DISC 2024) recently presented a uniform terminating leader election algorithm for *oriented* rings in this setting, with message complexity $O(n\mathsf{ID}_{\max})$ on a ring of size n, where $\mathsf{ID}_{\max}$ is the largest identifier in the system.

In this paper, we investigate the message complexity of leader election in this model, showing that no uniform algorithm can solve the problem if each process is limited to sending a constant number of messages in one direction.

Interestingly, this limitation hinges on the *uniformity* assumption. In the *non-uniform* setting – where processes know an upper bound $U \geq n$ on the ring size – we present an algorithm with message complexity $O(nU\mathsf{ID}_{\min})$, in which each process sends $O(U\mathsf{ID}_{\min})$ messages clockwise and only three messages counter-clockwise. Here, $\mathsf{ID}_{\min}$ is the smallest identifier in the system. This dependence on the identifiers compares favorably with the dependence on $\mathsf{ID}_{\max}$ of Frei et al. (DISC 2024).

We also show a non-uniform algorithm where each process sends $O(U \log \mathsf{ID}_{\min})$ messages in one direction and $O(\log \mathsf{ID}_{\min})$ in the other. The factor $\log \mathsf{ID}_{\min}$ is *optimal*, matching the lower bound of Frei et al. (DISC 2024).

Finally, in the anonymous setting, we propose a randomized algorithm where each process sends only $O(\log^2 U)$ messages, with a success probability of $1 - U^{-c}$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Content-Oblivious Networks, Leader Election, Oriented Rings, Asynchronous Systems

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.51

Related Version *Full Version*: https://arxiv.org/abs/2509.19187

Funding This work has been partially supported by ANR project DUCAT (ANR-20-CE48-0006), by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (24-1323-A0001), and by the Italian MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU through projects SERICS (PE00000014).



© Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The field of reliable distributed computing is characterized by the study of a wide range of failure models, from failures affecting specific processes [4, 17] to failures affecting communication channels, such as message omission failures, message addition failures or message corruptions [19]. In a recent paper, Censor-Hillel et al. [7] introduced a new interesting failure model for channels: the fully-defective asynchronous network. In this model, processes do not fail, but all messages in transit may be arbitrarily corrupted by an adversary. While the adversary can alter the content of any message, it cannot create or destroy messages.

This model is content-oblivious, meaning that messages carry no reliable content, not even the identity of the sender. The only information a message conveys is its existence.

Surprisingly, Censor-Hillel et al. [7] showed that if a leader process is initially known and the topology is 2-edge-connected, it is possible to simulate a reliable, uncorrupted message-passing network, even in asynchronous settings, within this fully-defective model.

This result makes studying leader election algorithms for this model particularly compelling: if a leader can be elected in a certain topology, with an algorithm that is terminating and where it is possible to distinguish in some way the messages of the leader election algorithm, then on such topology any asynchronous algorithm can be simulated even if all messages are arbitrarily corrupted. In this regard, Censor-Hillel et al. [7] conjectured that electing a leader was impossible in content-oblivious model and thus that having a leader was a necessary requirement. We stress that electing a leader is a fundamental problem in the classical message-passing setting, and it has been extensively studied in systems with unique identifiers [1, 12, 16, 18] and in anonymous networks [2, 3, 8, 21].

Frei et al. [13] disproved the aforementioned conjecture for a notable family of 2-edge-connected networks: oriented ring topologies. In an oriented ring, processes have a common notion of clockwise and counter-clockwise direction. More specifically, Frei et al. [13] proposed a quiescently terminating leader election algorithm with a message complexity of $O(n\mathsf{ID}_{\max})$, where $\mathsf{ID}_{\max}$ is the maximum identifier in the system. The quiescent termination of the algorithm allows it to be composed with the simulator of Censor-Hillel et al. [7] and thus implies that the presence of unique identifiers is enough to simulate any asynchronous algorithm on a content-oblivious oriented ring.

Additionally, Frei et al. [13] proved that, for any uniform algorithm, there exists an identifier assignment under which the algorithm must send $\Omega\left(n \log \frac{\mathsf{ID}_{\max}}{n}\right)$ messages. Although originally established for the uniform setting, this lower bound can be extended to the non-uniform setting, even when the *exact* value of n is known; see the full version of the paper for details.

In anonymous systems, where processes lack identifiers, Frei et al. [13] showed a randomized leader election algorithm using $n^{O(c^2)}$ messages and succeeding with probability $1 - n^{-c}$, but without explicit termination. In fact, Itai and Rodeh [15] proved that no uniform algorithm can both elect a leader and guarantee explicit termination in anonymous systems without additional assumptions. We stress that the lack of explicit termination does not allow the composition of this algorithm with the simulator of Censor-Hillel et al. [7].

Finally, we want to mention the related line of research of the beeping model [5, 6, 9, 20], where in each round a process may either emit a beep to its neighbors or listen. This model is inherently synchronous. Therefore, the techniques developed for leader election in the beeping model [10, 11, 14] cannot be applied in content-oblivious networks.

1.1 Contributions

We shed new light on the message complexity of leader election in the content-oblivious model. Specifically, we show that a uniform algorithm, even on oriented rings, cannot elect a leader while sending only a constant number of messages in a single direction.

Interestingly, we show that non-uniform algorithms, where processes know an upper bound U on the network size n, can circumvent this impossibility result.

In particular, we present an algorithm for oriented rings (Section 3) that elects a leader by sending $O(nU\mathsf{ID}_{\min})$ messages in the clockwise direction and where each process sends just three messages in the counter-clockwise direction. Here ID_{min} is the minimum identifier in the system. We also present a non-uniform algorithm for oriented rings (Section 4) that sends $O(nU \log \mathsf{ID}_{\min})$ messages in one direction and $O(n \log \mathsf{ID}_{\min})$ messages in the other. We stress that the factor $\log \mathsf{ID}_{\min}$ of our algorithm is *optimal*. This follows from the aforementioned lower bound of Frei et al. [13], which shows the necessity of logarithmic dependency on the values of the identifiers. Our result implies that $\Theta(\log \mathsf{ID}_{\min})$ is the *tight* message complexity for constant-size instances.

The key technical novelty in our non-uniform algorithms is the design of a global synchronization mechanism that leverages the known upper bound on the network size to keep processes partially synchronized across algorithmic phases. This mechanism enables processes to compare their identifiers bit by bit, leading to an algorithm whose complexity depends only on the size of the identifiers (and not on their values).

Regarding anonymous networks, with non-uniformity at hand, the aforementioned impossibility result of Itai and Rodeh [15] no longer applies. This allows us to create a randomized algorithm (Section 5) that, with probability at least $1 - U^{-c}$, is quiescently terminating. The algorithm has message complexity $O(n \log^2 U)$. In contrast to the randomized approach of Frei et al. [13], whose message complexity has an inherent exponential dependence on c, our algorithm achieves near-linear message complexity with only a multiplicative dependence on c, which is hidden in the $O(\cdot)$ notation.

For space reasons, in this brief announcement, we just announce our results and describe the high-level ideas; see the full version of this paper for technical details.

2 System Model

We consider a system composed of a set of n processes $P = \{p_0, p_1, \ldots, p_{n-1}\}$ that communicate on a ring network by sending messages to each other. More precisely, a process p has two local communication ports: port 0 and port 1. By means of a specific port, a process is able to send messages to and receive messages from one of its neighbors.

When the system is not anonymous, processes have unique identifiers that are arbitrarily selected from $\mathbb{N}$.

We assume an *asynchronous system*; that is, each message has an unpredictable delay that is finite but unbounded. A message cannot be lost. Messages are buffered, in the sense that if several messages are delivered to a process at the same time, they are stored in a local buffer and can then be retrieved by the destination process at its discretion. The local computation time at each process may be arbitrary but bounded, and processes do not share a notion of common time.

Processes are correct and do not experience any kind of failure.

Content-Oblivious Algorithms. Since we are interested in *content-oblivious algorithms*, we assume that messages do not carry any information apart from their existence. That is, each message is an empty string (a *pulse* [13]) and a process only observes the local port number on which the message is received.

Oriented Rings. We consider *oriented* rings. A ring is oriented if processes share a common notion of *clockwise* (CW) and *counter-clockwise* (CCW) orientation. More precisely, consider a ring $(p_0, p_1, \ldots, p_{n-1})$ for each $j \in [0, n-1]$, at process p_j, port 0 leads to p_{j+1} and port 1 leads to p_{j-1} (where indices are taken modulo n). We will say that a message is traveling in the clockwise direction if it is sent on port 0 and received on port 1 (i.e., a message goes from p_i to p_{i+1}); conversely, a message travels in the counter-clockwise direction if it is sent on port 1 and received on port 0.

Uniform and Non-Uniform Algorithms. An algorithm is *uniform* if it works for all possible network sizes; i.e., processes do not know the total number of processes n or a bound on it. An algorithm is *non-uniform* if processes know an upper bound $U \geq n$.

Message Complexity. We measure the performance of our algorithms by their *message complexity*. That is the number of all messages sent. We further discriminate between the numbers of clockwise messages and counter-clockwise messages sent by each process.

Leader Election Problem. In this problem, processes must elect one of them as the *leader* and all others are designated as *non-leader* processes. We consider *terminating leader election*, in the sense that every process must eventually enter either a `Leader` or `Non-Leader` state, and both of these states are final – meaning no further state changes are possible, and the process terminates the execution of the algorithm.

Moreover, we require that once the last process enters its final state, there are no messages in transit in the network. This is known as *quiescent termination* [13]. In our algorithms, the last process to enter its final state is always the unique leader. This property enables easy composability of algorithms: once the leader enters the final state, it knows that the network is "at rest" and can begin sending messages for a subsequent algorithm [13, Section 1.1].

3 Constant Number of Messages in One Direction

We now describe an algorithm where each process sends $O(U\mathsf{ID}_{\min})$ messages in the clockwise direction, and just 3 messages in the counter-clockwise direction. This algorithm is of interest as it shows that our claim below needs the uniformity assumption:

▶ **Theorem 1.** *For any uniform leader election algorithm and any bound $b \in \mathbb{N}$ there exists an oriented ring where at least one process sends more than b messages in each direction.*

The Constant Direction Algorithm, pseudocode in Algorithm 1, starts in a competing phase. In the competing phase, blue lines, a process p with identifier ID performs a loop of $U\mathsf{ID}$ iterations. In each iteration, the process sends a message in the CW direction and waits for a message from the CW direction. This phase synchronizes the processes so that the indices of any two processes in the phase loop can differ by at most $n-1$. The process with the minimum identifier will be the only one to end the competing phase by completing the prescribed number of iterations.

Algorithm 1 Constant Direction Algorithm(ID,boundOnSize).

```
1  for i ← 1 to boundOnSize ∗ ID do
2      send a message on port 0
3      receive a message on port q      // Wait for a message and store the port number in
         variable q
4      if q = 0 then break
5  if q = 0 then
       /* if a process enters this branch, then it is not the process with minimum ID
          */
6      send a message on port 1
7      receive a message on port 1                    // Wait for a message on port 1.
8      repeat
9          receive a message on port q
10         send a message on port 1 − q
11     until received two messages on port 0
12     return Non-Leader
13 else
       /* only the process with minimum ID enters this branch                    */
14     send a message on port 1
15     receive a message on port 0
16     send a message on port 1
17     repeat
18         receive a message on port q
19         if q = 1 then
20             send a message on port 0
21     until q = 0
22     send a message on port 1
23     receive a message on port 0
24     return Leader
```

This process then enters the termination detection phase (green lines). In this phase, the process first sends a message CCW to inform all other processes that they are not leaders. Once this message returns, it sends a second CCW message while starting to relay CW messages; when this second message will be received back the process will know that all CW messages in the network are eliminated. Finally, once the second message is received, it sends a third and final CCW message to terminate the algorithm.

A process whose identifier is not the minimum will enter the relay phase (red lines) upon receiving and relaying a message from the CCW direction. In the relay phase, the process first waits to receive a CW message, in order to eliminate the CW message it sent during the competing phase, and afterwards it relays both CCW and CW messages it receives. A non-leader process terminates after receiving three CCW messages.

▶ **Theorem 2.** *There exists a quiescently terminating leader election algorithm for oriented rings in which each process sends $O(U\mathsf{ID}_{\min})$ messages clockwise and three messages counterclockwise, for a total of $O(nU\mathsf{ID}_{\min})$ messages.*

4 An algorithm that sends $O(nU \log \mathsf{ID}_{\min})$ messages

We now describe an algorithm in which each process sends $O(U \log \mathsf{ID}_{\min})$ messages. The algorithm uses an encoding of identifiers such that the smallest identifier is not a prefix of any other identifier, and the last bit of every encoded identifier is 0. To achieve this, given an identifier ID of bit length ℓ, we encode it as $1^\ell 0 \cdot \mathsf{ID} \cdot 0$.

The algorithm elects the process with the minimum identifier and proceeds in elimination rounds. In each round, active processes compare a specific bit of their (encoded) identifiers, eliminating all processes that do not have the minimum bit at that position.

The bound U on the network size is used to synchronize processes across different rounds of the algorithm. When an active process enters a round, it begins the *Synchronization phase*. In this phase, each active process sends and receives messages in the CW direction for U steps. Then, processes with bit 0 send and receive one message in the CCW direction, blocking messages in the CW direction (this is the *Zero Signaling phase*). Meanwhile, processes with bit 1 attempt to continue sending and receiving messages in the CW direction for another U steps. However, they become inactive if they receive a CCW message, indicating the presence of a process with bit 0 (this is the *No-Zero Checking phase*).

The rationale for these phases is to ensure that processes with bit 1 proceed to the next round in the active state if and only if no process with bit 0 exists in the current round. Indeed, when processes with bit 0 stop propagating CW messages, the processes with bit 1 are prevented from completing their *No-Zero Checking phase*.

Conversely, a process with bit 0 in a given round remains active. The *Synchronization phase* described above ensures that once an active process completes this phase for a new round, all active processes have completed the previous round and are in the *Synchronization phase* of the current round.

An inactive process relays messages, and while doing so, it eliminates the first CW message it receives. This guarantees that, once all inactive processes have eliminated one message, the total number of messages in the network equals the number of active processes. Therefore, once all processes except the one with the minimum identifier have been eliminated, only a single message remains in the network.

The algorithm terminates after $|\mathsf{ID}_{\min}| = O(\log \mathsf{ID}_{\min})$ rounds. At this point, the only remaining active process is the one with the minimum identifier. This process can detect this locally and communicates it to all other processes (which are, by construction, inactive) by sending an additional message in the CCW direction (this is the *Termination phase*).

All other processes detect termination when they receive two consecutive messages in the CCW direction: the first sent during the *Zero Signaling phase* by the process with the minimum identifier, and the second sent during the *Termination phase*. Therefore, we have the following theorem.

▶ **Theorem 3.** *There exists a quiescently terminating leader election algorithm for oriented rings in which each process sends $O(U \log \mathsf{ID}_{\min})$ messages clockwise and $O(\log \mathsf{ID}_{\min})$ messages counterclockwise, for a total of $O(nU \log \mathsf{ID}_{\min})$ messages.*

5 Randomized Leader Election

In our randomized algorithm, each process samples its ID uniformly at random from the set $\{0, 1, 2, \ldots, 2^{\lceil c_1 \log U \rceil} - 1\}$. With probability at least $1 - U^{2-c_1}$, all processes receive distinct identifiers. The bit-by-bit comparison algorithm from Section 4 is then executed using these random identifiers.

This sampling strategy yields a useful property: for any fixed bit index i, every maximal consecutive sequence of processes whose bit $\mathsf{ID}[i]$ equals 1 has length at most $\lceil c_2 \log U \rceil$ with probability at least $1 - U^{1-c_2}$. This property reduces the number of steps in the *Synchronization* and *No-Zero Checking* phases of the algorithm in Section 4 from U to $\lceil c_2 \log U \rceil$, thereby improving the message complexity and establishing our final theorem.

▶ **Theorem 4.** *For any constant $c > 0$, there exists a randomized quiescently terminating leader election algorithm for anonymous oriented rings with a success probability of $1 - U^{-c}$ in which each process sends $O(\log^2 U)$ messages clockwise and $O(\log U)$ messages counterclockwise, for a total of $O(n \log^2 U)$ messages.*

References

1 H. Attiya and J. L. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley, 2004.

2 P. Boldi, S. Shammah, S. Vigna, B. Codenotti, P. Gemmell, and J. Simon. Symmetry breaking in anonymous networks: Characterizations. In *ISTCS 1996*, pages 16–26, 1996.

3 P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1):21–66, 2002. doi:10.1016/S0012-365X(00)00455-6.

4 C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

5 A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Counting in one-hop beeping networks. *Theoretical Computer Science*, 780:20–28, 2019. doi:10.1016/J.TCS.2019.02.009.

6 A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Design patterns in beeping algorithms: Examples, emulation, and analysis. *Information and Computation*, 264:32–51, 2019. doi:10.1016/J.IC.2018.10.001.

7 K. Censor-Hillel, S. Cohen, R. Gelles, and G. Sela. Distributed computations in fully-defective networks. *Distributed Computing*, 36(4):501–528, 2023. doi:10.1007/S00446-023-00452-2.

8 J. Chalopin, E. Godard, and Y. Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Computing*, 25(4):297–311, 2012. doi:10.1007/S00446-012-0163-Y.

9 A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *DISC 2010*, volume 6343 of *Lecture Notes in Comput. Sci.*, pages 148–162. Springer, 2010. doi:10.1007/978-3-642-15763-9_15.

10 A. Czumaj and P. Davies. Leader election in multi-hop radio networks. *Theoretical Computer Science*, 792:2–11, 2019. doi:10.1016/J.TCS.2019.02.027.

11 F. Dufoulon, J. Burman, and J. Beauquier. Beeping a deterministic time-optimal leader election. In *DISC 2018*, volume 121 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.DISC.2018.20.

12 P. Flocchini, E. Kranakis, D. Krizanc, F. L. Luccio, and N. Santoro. Sorting and election in anonymous asynchronous rings. *Journal of Parallel and Distributed Computing*, 64(2):254–265, 2004. doi:10.1016/J.JPDC.2003.11.007.

13 F. Frei, R. Gelles, A. Ghazy, and A. Nolin. Content-oblivious leader election on rings. In *DISC 2024*, volume 319, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.DISC.2024.26.

14 K.-T. Förster, J. Seidel, and R. Wattenhofer. Deterministic leader election in multi-hop beeping networks. In *DISC 2014*, volume 8784 of *Lecture Notes in Comput. Sci.*, pages 212–226. Springer, 2014.

15 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Inf. Comput.*, 88(1):60–87, July 1990. doi:10.1016/0890-5401(90)90004-2.

16 S. Kutten, W. K. Moses Jr., G. Pandurangan, and D. Peleg. Singularly optimal randomized leader election. In *DISC 2020*, volume 179 of *LIPIcs*, pages 22:1–22:18, 2020. doi:10.4230/LIPICS.DISC.2020.22.

17 M. Raynal. *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. Springer, 2018.

18 N. Santoro. *Design and Analysis of Distributed Algorithms*. John Wiley & Sons, 2007.

19 N. Santoro and P. Widmayer. Time is not a healer. In *STACS 1989*, volume 349 of *Lecture Notes in Comput. Sci.*, pages 304–313. Springer, 1989. doi:10.1007/BFB0028994.

20 R. Vacus and I Ziccardi. Minimalist leader election under weak communication. In *PODC 2025*. ACM, 2025.

21 M. Yamashita and T. Kameda. Computing on anonymous networks: Part I - Characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996. doi:10.1109/71.481599.

Brief Announcement: From Few to Many Faults: Adaptive Byzantine Agreement with Optimal Communication

Andrei Constantinescu
ETH Zürich, Switzerland
DFINITY Foundation, Zürich, Switzerland

Marc Dufay
ETH Zürich, Switzerland

Anton Paramonov
ETH Zürich, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

We study the problem of Strong Byzantine Agreement and establish tight upper and lower bounds on communication complexity, parameterized by the actual number of Byzantine faults. Specifically, for a system of n parties tolerating up to t Byzantine faults, out of which only $f \leq t$ are actually faulty, we obtain the following results:

In the partially synchronous setting, we present the first Byzantine Agreement protocol that achieves *adaptive* communication complexity of $\mathcal{O}(n + t \cdot f)$ words, which is asymptotically optimal. Our protocol has an optimal resilience of $t < n/3$.

In the asynchronous setting, we prove a lower bound of $\Omega(n + t^2)$ on the expected number of messages, and design an almost matching protocol with an optimal resilience that solves agreement with $\mathcal{O}((n + t^2) \cdot \log n)$ words. Our main technical contribution in the asynchronous setting is the utilization of a bipartite expander graph that allows for low-cost information dissemination.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine Agreement, Communication Complexity, Adaptive Communication Complexity, Resilience

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.52

Related Version *Full Version*: arXiv:2505.19989 [10]

1 Introduction

Achieving agreement in a distributed setting is fundamental to blockchain systems. Modern blockchains often comprise thousands of nodes attempting to reach consensus. Given the widely accepted resilience model where $t \approx n/3$ validators can be Byzantine, and considering the classical Dolev-Reischuk lower bound of $\Omega(n^2)$ messages for agreement [11], such systems must exchange millions of messages. This communication overhead results in high latency and limited scalability. This work addresses this challenge through two key contributions:

We bypass the Dolev-Reischuk lower bound by introducing the first partial-synchrony algorithm for Byzantine Agreement that requires only $\mathcal{O}(n + t \cdot f)$ messages. This approach is significantly faster compared to the $\mathcal{O}(n^2)$ suggested by [11], since often in practice the actual number of misbehaving nodes is small. Our algorithm achieves asymptotically optimal communication complexity, as well as optimal resilience.

Furthermore, we highlight that system robustness is determined not by the relative fraction of possible corruptions but by the absolute number of failures it can tolerate. For instance, it's easier for an adversary to corrupt a system with 3,001 nodes that tolerates


© Andrei Constantinescu, Marc Dufay, Anton Paramonov, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0



1,000 faults than one with 10,000 nodes tolerating 2,000 faults. Building on this insight, we design an algorithm that decouples n and t, achieving superior performance when $t \ll n$. Specifically, instead of the known $\mathcal{O}(n \cdot t)$ communication complexity, in the asynchronous setting, we achieve agreement with a communication complexity of $\mathcal{O}((n + t^2) \cdot \log n)$. This allows system designers to first choose a maximal threshold t of tolerated failures that would still allow for efficient dissemination of t^2 messages, and then scale n up to t^2 without introducing additional latency. Notably, our algorithm is non-constructive, since it internally utilizes a bipartite expander whose existence has been proven, but no explicit construction is known; when restricted to explicit constructions, the best-known guarantees are slightly weaker.

Finally, we establish that achieving adaptive communication complexity is infeasible in asynchronous settings. Specifically, we prove a lower bound of $\Omega(n + t^2)$ for any algorithm that guarantees almost-sure termination. This result demonstrates that our algorithm is optimal up to a logarithmic factor.

2 Related Work

Synchrony. In synchronous networks, deterministic Byzantine Broadcast (BB) and Weak Byzantine Agreement (BA) achieve cheap communication: Cohen et al. [9] give $\mathcal{O}(n(f + 1))$ for Weak BA/BB, and [7] give deterministic BA with $\mathcal{O}(nf)$ at resilience $t < (\frac{1}{2} - \varepsilon)n$. Civit et al. [5] extend to Multi-Valued BA (MVBA) and Interactive Consistency (IC) with bit complexity $\mathcal{O}(L_o n + n(f + 1)\kappa)$. Lower bounds include $\Omega(f^2)$ messages [11], $\Omega(n + tf)$ bits [15], and even randomized expected $\Omega(f^2)$ under adaptive adversaries [1].

Partial synchrony. In partial synchrony, BA attains $\mathcal{O}(n^2)$ communication complexity, e.g., Oper [6] Introduced by Civit et al., lifts any synchronous BA algorithm to partial synchrony while preserving per-process bit costs (yielding total $\mathcal{O}(n^2)$ for [5]). For State Machine Replication (SMR) in partial synchrony, HotStuff with robust view synchronization reaches $\mathcal{O}(nf)$ after GST [17, 14]. As for the lower bounds, Spiegelman [15] showed that for any partially synchronous algorithm, the number of messages sent before GST can be unbounded.

Asynchrony. In asynchrony, the classical FLP [12] impossibility rules out deterministic BA. However, randomness enables expected $\mathcal{O}(n^2)$ communication complexity to achieve an almost sure termination [4] and subquadratic $\mathcal{O}(n \log^2 n)$ for with high probability guarantees [8]. When allowing for an initial setup, expected $\mathcal{O}(n\kappa^4)$ can be achieved [2].

3 Preliminaries

We consider a set of n parties running a distributed protocol in a fully connected network. Channels are authenticated, meaning that when receiving a message, a party knows who sent it.

Network Setting. We consider 3 different network settings:

- *Synchronous network*: We assume there is a known $\Delta > 0$ such that when sending a message, a party has the guarantee that this message will be received within Δ units of time. This allows protocols to be expressed in rounds where each round takes Δ units of time.

Table 1 Comparison of Byzantine Agreement Protocols. When a communication is stated with Ω, it indicates a lower bound result.
$*$ - [14] uses random leader election, but can be derandomized, maintaining the mentioned characteristics.
§ - unlimited messages before GST., † - with high probability., †† - with overwhelming probability. , ‡ - with probability 1., E - in expectation.

Paper	Problem	Communication	Resilience	Det.	PKI	Properties
Synchrony						
Cohen et al. [9]	BB	$\mathcal{O}(n \cdot f)$	$t < n/2$	Yes	Yes	A;T;V
Civit et al. [7]	MVBA	$\mathcal{O}(n \cdot f)$	$t < (\frac{1}{2} - \varepsilon)n$	Yes	Yes	A;T;V
Civit et al. [5]	MVBA, IC	$\mathcal{O}(L_o n + n \cdot f)$	$t < \frac{n}{2}$	Yes	Yes	A;T;V;IC
This paper	BA	$\mathcal{O}(n + tf)$	$t < \frac{n}{2}$	Yes	Yes	A;T;V
Spiegelman et al. [15]	BA	$\Omega(n + tf)$	Any	Yes	No	A;T;V
Ittai et al.[1]	BA	$\mathbb{E}(\Omega(f^2))$	Any	No	Yes	A;T;V
Partial Synchrony						
Maofan et al. [17, 14]	SMR	$\mathcal{O}(n \cdot f)$	$t < n/3$	Yes*	Yes	A;T;EV
Civit et al. [6]	BA	$\mathcal{O}(n^2)$	$t < n/3$	Yes	No	A;T;V
This paper	BA	$\mathcal{O}(n + t \cdot f)$	$t < n/3$	Yes	Yes	A;T;V
Spigeleman et al. [15]	BA	$\Omega(\infty)^{\S}$	Any	Yes	No	A;T;V
Asynchrony						
Cachin et al. [4]	BA	$\mathbb{E}(\mathcal{O}(n^2))$	$t < n/3$	No	Yes	A;T‡;V
Cohen et al. [8]	BA	$\mathcal{O}(n \cdot \log^2 n)^{\dagger}$	$t < (\frac{1}{3} - \varepsilon)n$	No	Yes	A†;T†;V†
Blum et al. [2]	BA	$\mathbb{E}[\mathcal{O}(n \cdot \kappa^4)]$	$t < (\frac{1}{3} - \varepsilon)n$	No	Yes	A††;T††;V††
This paper	BA	$\mathbb{E}(\mathcal{O}((n + t^2) \cdot \log n))$	$t < n/3$	No	Yes	A;T‡;V
This paper	BA	$\mathbb{E}(\Omega(n + t^2))$	Any	No	Yes	A;T‡;V

- *Asynchronous network*: There is no guarantee on the delay between a party sending a message and it being received. The only guarantee is that a message sent is eventually received.
- *Partially synchronous network*: We assume the setting with a known $\Delta > 0$ and an unknown *Global Stabilization Time* (GST), such that a messages sent at time t is received by the time at most $\max\{t, GST\} + \Delta$.

Byzantine Behavior. We assume that a set of at most t parties can become faulty when running the protocol. In particular, an adversary is able to control these parties, and the protocol must be resilient to this setting. Parties that are not corrupted are called honest. In the rest of this paper, we will also consider f, the number of actual byzantine parties during an execution of a protocol, we have $0 \leq f \leq t \leq n$.

Moreover, we assume that for the asynchronous setting as well as the partially synchronous setting before GST, the adversary has full control over the scheduler and can delay or reorder any messages as long as they are eventually received. Our protocols are resilient against a dynamic adversary. This means that the adversary is allowed to observe messages being sent before deciding which parties to corrupt.

Byzantine Agreement. In this paper, we focus on solving the problem of Strong Binary Byzantine Agreement, which we call just Byzantine Agreement for simplicity. In this problem, each honest party can *propose* a value (either 0 or 1) and can *decide* a value. The Byzantine Agreement is defined by the following properties.

- *Termination*: Every honest party eventually decides a value.

- *Agreement*: If two honest parties p_1 and p_2 respectively decide values v_1 and v_2, then $v_1 = v_2$.
- *Strong Unanimity*[1]: If all honest parties propose the same value, then this value must be decided.
- *Probabilistic termination*: Every honest party eventually decides a value almost surely.

For the synchronous and partially synchronous setting, we say that a protocol satisfies Byzantine Agreement (BA) if it satisfies Termination, Agreement, and Strong Unanimity. Because of the FLP impossibility result [12] BA cannot be solved deterministically in asynchrony. Therefore, for the asynchronous setting, we replace *Termination* with *Probabilistic Termination*:

Cryptographic Primitive. In some of our protocols, we will assume the presence of a public-key infrastructure (PKI). This allows for *threshold* signature schemes [3] which are used for combining multiple signatures into one and hence reducing the communication complexity of the algorithm.

Communication Complexity. We define the *communication complexity* of an algorithm as the number of *words* sent by honest nodes, where each word contains a constant number of signatures and a constant number of bits. As pointed out by Spiegelman et al. [15], there doesn't exist a partially synchronous algorithm that solves BA sending a bounded number of messages if messages sent before GST are counted. Therefore, when speaking about the communication complexity of partially synchronous algorithms, we refer only to messages sent *after* GST.

4 Results Overview

In this section, we formally state our results and describe the main components used to obtain them. On the feasibility side, we devise protocols with adaptive communication complexity for all major time models.

In partial synchrony, our algorithm achieves asymptotically optimal communication complexity.

▶ **Theorem 1.** *There exists a deterministic algorithm that tolerates up to $t < n/3$ Byzantine faults and, given a PKI, solves Byzantine Agreement in partial synchrony with communication complexity of $\mathcal{O}(n + t \cdot f)$.*

In an asynchronous setting, we show the existence of an algorithm that achieves optimal resilience and a communication complexity that is asymptotically optimal up to a logarithmic factor.

▶ **Theorem 2.** *There exists a randomized algorithm that tolerates up to $t < n/3$ Byzantine faults and, given a PKI, solves Byzantine Agreement in asynchrony with expected communication complexity of $O((n + t^2) \cdot \log n)$.*

In synchrony, our algorithm has asymptotically optimal adaptive communication and optimal resilience.

[1] In literature, this property is also called Strong Validity.

▶ **Theorem 3.** *There exists a deterministic algorithm that tolerates up to $t < n/2$ Byzantine faults and, given a PKI, solves Byzantine Agreement in synchrony with communication complexity of $\mathcal{O}(n + t \cdot f)$.*

All of our results follow a common high-level strategy. First, we execute an adaptive Byzantine Agreement protocol on a subset of parties of size $\Theta(t)$, which we refer to as the *quorum*. The communication complexity here only depends on t and f. We note that for this step we rely on existing Byzantine Agreement protocols in the synchronous and asynchronous settings, specifically those of Civit et al.[5] and Cachin et al.[4]. In contrast, the $\mathcal{O}(n + n \cdot f)$ protocol we use for the partially synchronous setting is our own contribution, described in Section 5. While inspired by Spiegelman's synchronous protocol for external validity [15], our algorithm advances it in two key ways: it operates in the partially synchronous setting and guarantees strong unanimity.

In the second step, we execute a Quorum-to-All Broadcast (QAB) – a communication primitive introduced in this work – where the quorum nodes disseminate the decided value to the rest of the network. We design QAB protocols for both partially synchronous and asynchronous settings. In particular, our asynchronous QAB leverages bipartite expander graphs to enable efficient communication, constituting one of the key technical contributions of this paper.

On the infeasibility side, we have the following versatile lower bound that shows the impossibility of achieving an adaptive communication complexity in the asynchronous setting.

▶ **Theorem 4.** *Let $\mathcal{A}$ be a protocol solving asynchronous byzantine agreement resilient to t byzantine parties. $\mathcal{A}$ is allowed to have access to PKI, shared randomness, and an initial setup phase before receiving values. Let M be the number of messages exchanged after setup to reach agreement, then there exists an input configuration and a message scheduling protocol such that $\mathbb{E}(M) = \Omega(t^2)$ without any byzantine party. This lower bound holds even if the adversary is static.*

The main contribution of this result – beyond the fact that it holds under a very weak adversary and a strong algorithmic setting – is that, to the best of our knowledge, it establishes the first lower bound of $\Omega(t^2)$, in contrast to the $\Omega(f^2)$ bounds shown in [11, 1]. This bound allows us to claim that the algorithm stated by Theorem 2 has an almost optimal communication complexity, since $\Omega(n)$ is a trivial lower bound.

5 Adaptive Byzantine Agreement in Partial Synchrony

In this section, we present a partially synchronous algorithm that achieves an adaptive communication complexity of $\mathcal{O}(n + n \cdot f)$. Our algorithm is based on a modified version of Spiegelman's byzantine agreement protocol for external validity [15], which itself uses Hotstuff's view synchronization [17] technique. While inspired by Spiegelman's protocol for network-agnostic settings [15], our work departs significantly in both design and guarantees. We eliminate the asynchronous and randomized components of the original protocol and instead strengthen the synchronous part with additional checks, enabling it to function effectively in partial synchrony after GST. This modification allows us to achieve $\mathcal{O}(n \cdot f)$ message complexity for external validity without relying on full synchrony. Crucially, by allowing parties to selectively share their commit proofs upon request, we ensure termination without incurring additional message overhead. Moreover, we extend the protocol to achieve binary strong unanimity – rather than just external validity – by leveraging threshold signatures.

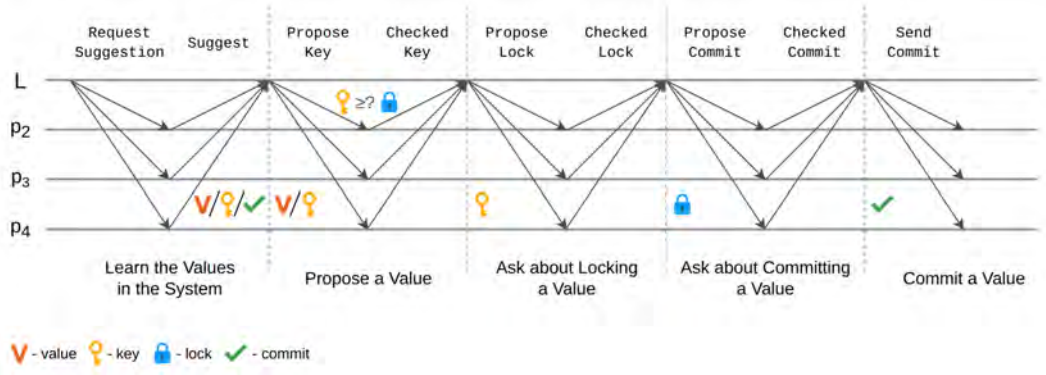


Figure 1 Evolution of a single view in our partially synchronous Byzantine Agreement protocol, assuming an honest leader L and that the system is after GST.

The protocol is organized in views. Each view has a leader, who is chosen in a round-robin way. In the full version of the paper [10], we show that the algorithm is guaranteed to succeed after at most n views after GST. Moreover, if there are no byzantine parties, everyone is guaranteed to decide within constant time after GST.

The algorithm for a single view consists of 4 phases of a leader asking and other parties responding. A leader proceeds to the next phase once they get $n - t$ responses. On each phase, the leader tries to acquire a new threshold signature, namely the key signature, then the lock and finally the commit signature, each with stronger properties than the other. After 4 phases pass successfully, the leader obtains broadcasts his value and is guaranteed than only this value can be decided in the future (see Figure 1). We note that if the leader is byzantine or the network is not yet synchronous, the last phase may not be reached by the end of the view. Even then, the threshold values obtained so far are used in subsequent rounds to ensure agreement. A complete description of our protocol as well as a detailed proof for it is given in the full version of our paper [10].

This algorithm, coupled with the QAB primitive, solves Byzantine agreement while tolerating up to $t < n/3$ parties and using $\mathcal{O}(n + t \cdot f)$ messages. Therefore, it satisfies Theorem 1.

6 Broadcasting from a Quorum

To decrease the message complexity, we will rely on a primitive which we call Quorum-to-All Broadcast (abbreviated QAB). Briefly, this primitive allows a small group (of size $\Theta(t)$) of parties all having the same input value v_{in} to broadcast it to everyone. QAB is defined by the following property that encapsulates classical Agreement, Termination, and Validity.

- *Complete correctness:* Every honest node eventually decides v_{in}.

We next describe our QAB for Asynchrony at a high level. For formal description, as well as the partial synchrony QAB, please refer to the full version of the paper [10].

▶ **Theorem 5.** *For a Quorum of size $3t + 1$, there exists a deterministic algorithm that, given a PKI, solves QAB in asynchrony in the presence of at most t Byzantine faults with communication complexity of $\mathcal{O}((n + t^2) \cdot \log n)$.*

In order to achieve the stated communication complexity, we restrict the communication between nodes to a specific graph, which we call a communication graph. There are three roles in the algorithm (one node can have multiple roles): *quorum nodes* - nodes that have a

common value to broadcast, *relayers* - nodes that help quorum nodes disseminate the value and *parties* - all the nodes, those that need to learn the value. First, quorum nodes agree on a value based on their own proposals, then disseminate it through relayers. Parties then confirm that they have received the value by sending a signed confirmation to relayers who in turn aggregate these confirmations and relay them to quorum nodes. If a quorum node does not receive a confirmation from some party, it sends a value to it directly.

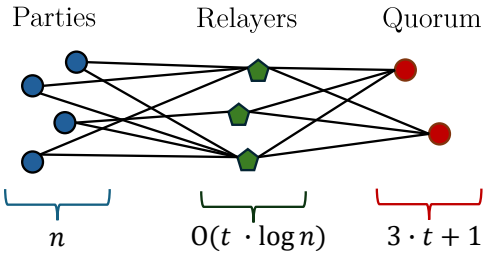


Figure 2 The communication graph for stages 1 and 2 of the algorithm. Nodes are assigned three roles (non-exclusive): *parties*, *relayers* and *quorum* with respective sizes of n, $O(t \log n)$ and $3t + 1$. Each party is only linked to $O(\log n)$ relayers, and each relayer is linked to each quorum node.

The crux of our result is the utilization of a bipartite (*parties, relayers*) graph that (I) has few edges (II) an adversary cannot disconnect many parties from the quorum. Please see the full paper [10] for a formal statement of its properties.

We remark that the existence of a bipartite expander with desired properties is only shown non-constructively in [16]. Known explicit bipartite expanders with polynomial time construction [13], when utilized instead, would imply a message complexity of $(n + t^{2+\varepsilon}) \cdot \log^{\mathcal{O}(1/\varepsilon)} n$.

References

1 Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 317–326. ACM, July / August 2019. doi:10.1145/3293611.3331629.

2 Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Cham, November 2020. doi:10.1007/978-3-030-64375-1_13.

3 Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Berlin, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_26.

4 Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM PODC*, pages 123–132. ACM, July 2000. doi:10.1145/343477.343531.

5 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM PODC*, pages 145–156. ACM, June 2024. doi:10.1145/3662158.3662792.

6 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Manuel Vidigueira, and Igor Zablotchi. Partial synchrony for free: New upper bounds for

byzantine agreement. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4227–4291. SIAM, 2025. doi:10.1137/1.9781611978322.144.

7 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Strong byzantine agreement with adaptive word complexity, 2023. doi:10.48550/arXiv.2308.03524.

8 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In *34th International Symposium on Distributed Computing (DISC 2020)*, pages 25:1–25:17, 2020. doi:10.4230/LIPIcs.DISC.2020.25.

9 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Brief announcement: Make every word count: Adaptive byzantine agreement with fewer words. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 421–423. ACM, July 2022. doi:10.1145/3519270.3538458.

10 Andrei Constantinescu, Marc Dufay, Anton Paramonov, and Roger Wattenhofer. From few to many faults: Adaptive byzantine agreement with optimal communication, 2025. Full version of this paper. doi:10.48550/arXiv.2505.19989.

11 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. In Robert L. Probert, Michael J. Fischer, and Nicola Santoro, editors, *1st ACM PODC*, pages 132–140. ACM, August 1982. doi:10.1145/800220.806690.

12 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

13 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *Journal of the ACM (JACM)*, 56(4):1–34, 2009. doi:10.1145/1538902.1538904.

14 Andrew Lewis-Pye and Ittai Abraham. Fever: optimal responsive view synchronisation. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, pages 14:1–14:16, 2024. doi:10.4230/LIPIcs.OPODIS.2023.14.

15 Alexander Spiegelman. In Search for an Optimal Authenticated Byzantine Agreement. In *35th International Symposium on Distributed Computing (DISC 2021)*, pages 38:1–38:19, 2021. doi:10.4230/LIPIcs.DISC.2021.38.

16 Ola Svensson. Lecture 3: Bipartite expander graphs. https://theory.epfl.ch/courses/topicstcs/Lecture3.pdf, 2021. Accessed: 2025-04-16.

17 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 347–356. ACM, July / August 2019. doi:10.1145/3293611.3331591.

Brief Announcement: Maintaining a Bounded Degree Expander in Dynamic Peer-To-Peer Networks

Antonio Cruciani
Aalto University, Espoo, Finland

Abstract

We study the problem of maintaining robust and sparse overlay networks in fully distributed settings where nodes continuously join and leave the system. This scenario closely models real-world unstructured peer-to-peer networks, where maintaining a well-connected yet low-degree communication graph is crucial. We generalize a recent protocol by Becchetti et al. [SODA 2020] that relies on a simple randomized connection strategy to build an expander topology with high probability to a dynamic networks with churn setting. In this work, the network dynamism is governed by an oblivious adversary that controls which nodes join and leave the system in each round. The adversary has full knowledge of the system and unbounded computational power, but cannot see the random choices made by the protocol. Our analysis builds on the framework of Augustine et al. [FOCS 2015], and shows that our distributed algorithm maintains a constant-degree expander graph with high probability, despite a continuous adversarial churn with a rate of up to $\mathcal{O}(n/polylog(n))$ per round, where n is the stable network size. The protocol and proof techniques are not new, but together they resolve a specific open problem raised in prior work. The result is a simple, fully distributed, and churn-resilient protocol with provable guarantees that align with observed empirical behavior.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Networks → Peer-to-peer networks; Theory of computation → Expander graphs and randomness extractors

Keywords and phrases Peer-to-peer network, dynamic network, churn, distributed algorithm, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.53

Related Version *Full Version*: `https://arxiv.org/abs/2506.17757` [6]

Funding *Antonio Cruciani*: This work was supported in part by the Research Council of Finland, Grant 363558.

1 Introduction

In this brief announcement, we study a simple *dynamics* that builds and maintains sparse and well connected graph despite an *adversarial churn* at each round.

As a motivating example, consider Peer-to-Peer (P2P) networks that are (for example) at the base of major blockchain systems such as Bitcoin and Ethereum in which the system provides decentralized communication and resilience against single points of failure. These overlays are governed by strict privacy rules restricting any leak of sensitive information about the peers. Compliance to such privacy rules essentially makes it difficult to employ any distributed protocol that explicitly uses standard message passing techniques. For example (say in the P2P Bitcoin Network), it is not possible to use random walks to "uniformly spread" the node IDs (in this case IP Addresses) on the network and provide peers with a fresh set of uniform IDs from which to sample new neighbors. Indeed, after an initial bootstrap phase in which the nodes rely on DNS seeds for node discovery, nodes running the Bitcoin-core implementation turn to a decentralized policy to rebuild their neighborhood


© Antonio Cruciani;
licensed under Creative Commons License CC-BY 4.0



when their degree drops below the configured threshold. Each node has a minimum and a maximum number of neighbors that they must have (respectively 8 and 125, in the default configuration) and it locally stores a large list of IP addresses of active nodes. Every time the number of current neighbors of a node drops below the configured minimum value, it tries to find new connections with nodes sampled from its list. Such a list, is periodically shared to its neighbors and updated with the lists received by the neighbors. In the long run each node samples its neighbors from a list that represent a sufficiently random subset of nodes of the network. This sampling-based mechanism, despite being decentralized and oblivious to the global structure of the network, enables the construction of a sparse and robust overlay that induce good expansion properties. In particular, Becchetti et al. [5] formalized and analyzed a dynamic random graph model inspired by the Bitcoin protocol, showing that the resulting network converges rapidly to a constant-degree expander with high probability.

However, real-world P2P networks are inherently dynamic: nodes frequently join and leave the system over time. In this brief announcement, we address an open question posed by Becchetti et al. [5] by adapting their RAES protocol to a dynamic setting with adversarial churn. Our solution builds directly on the analytical framework developed by Augustine et al. [2], and shows that a RAES-style protocol can maintain expansion with high probability even under continuous adversarial churn. This work serves as a theoretical closure to earlier experimental studies of RAES variants under churn [8]. While the protocol and techniques are not original–both are adapted from prior work–the analysis formally confirms the empirical behavior observed in simulations. The result is a modest but rigorous endpoint to a research line focused on the resilience of randomized overlay networks.

2 Preliminaries

A *dynamic graph*[1] $\mathcal{G}$ is a family of graphs $\mathcal{G} = \{G_t = (V_t, E_t) : t \in \mathbb{N}\}$. We call G_t the *snapshot* of the dynamic graph at time t. For a set of nodes $S \subseteq V_t$, the *edge-expansion* of the snapshot G_t is defined as the minimum over all set of nodes $S \subseteq V_t$ whose size is $0 < |S| \leq n/2$ of the ratio between the number of edges leaving the set and its size.

The Dynamic Network with Churn Model. We consider a synchronous dynamic network controlled by an *oblivious* adversary. The adversary fixes a sequence of graphs $\mathcal{H} = (H_1, H_2, \dots, H_t, \dots)$ with each H_t defined on a vertex set V_t. The size of the vertex set is assumed to be stable. In other words, we have $|V_t| = n$ for all t. Furthermore, we assume that $|V_t \setminus V_{t+1}| = |V_{t+1} \setminus V_t| \in \mathcal{O}(n/\log^k n)$ where k is a constant determined in the analysis. The node set V_t is the set of nodes present in the network at round t. We assume that each node has a unique ID chosen from an ID space of size poly(n). The degree of each graph H_t is bounded by a constant δ. For the first $B = polylog(n)$ rounds, the adversary is *silent*, i.e., there is no churn, more precisely $H_1 = H_2 = \cdots = H_B$. Such an initial period is called *bootstrap phase*, and we can think of this phase as a period of stability during which the protocol builds and initial bounded degree expander graph. Subsequently, the network is said to be in the *maintenance phase* during which it can experience churn in the sense that a large number of nodes might join and leave dynamically at each step. During the maintenance phase, we require that any new node v that enters the network at round t must be connected in H_t to at least one pre-existing node v, i.e., if $u \in V_t \setminus V_{t-1}$, then u must be connected to some $v \in V_{t-1} \cap V_t$ in round t. The adversary must ensure that the degree

[1] In this paper we will use dynamic-graph and dynamic-network interchangeably.

bound δ is not violated. We only require that a new node is connected to a pre-existing node in its very first round upon entering the network. Secondly, after the bootstrap phase we do not require H_t to be connected. Indeed, the only required edges in H_t are the edges connecting new nodes to pre-existing ones. This is a standard requirement in the Dynamic Network with Churn (DNC) model (see, for example, [3, 2]) and prevents the adversary to create disconnected clusters. Our goal is to design a simple protocol that maintains a graph process $\mathcal{G} = \{G_t = (V_t, E_t)\}_{t \geq 1}$ over time such that each G_t for $t \geq B$, with high probability, has good expansion while maintaining a constant degree bound for each node between d and Δ such that $d < \Delta$ are two constants. We assume that, at time t, all the nodes have the ability of sampling from a uniform distribution over the vertex set at time t.

Remark. Although the assumption that a node can pick its neighbors uniformly at random among all nodes of the network is unrealistic in many scenarios, the edge creation processes in our model is reminiscent of the way some unstructured peer-to-peer networks such as the Bitcoin Network maintain a "random" topology.

The RAES Protocol. *RAES (Request a link, then Accept if Enough Space)* [5] is a random graph model defined by three parameters $n \in \mathbb{N}, d \in \{1, \ldots, n-1\}, c > 1$, in which each one of n nodes has degree at least d and at most $cd = \Delta$. The random graph is generated according to the discrete random process described in Algorithm 1. The process terminates when all nodes have degree in $[d, \Delta]$.

Algorithm 1 Overview of the RAES protocol.

```
Let G = (V, {∅}).
foreach t ≥ 0 do
    Phase 1 (reconnection): Each node u with degree d(u) < d, picks d − d(u) new
      neighbors uniformly at random.
    Phase 2 (degree adjustment): Each node u with degree d(u) > Δ selects
      d(u) − Δ neighbors u uniformly at random and drops the connection with them.
end
```

The RAES model can be seen as a simplified version of the network-formation process implemented in Bitcoin-core [9, 1]. The protocol converges in $\mathcal{O}(\log n)$ rounds to an expander graph with high probability [5]. However, this is not guaranteed to happen when nodes continuously join and leave the network.

3 Our Contribution

We address the problem of defining a variation of the RAES protocol that can build and maintain a dynamic expander graph despite an adversarial churn of $\mathcal{O}(n/\log^k n)$ at each round. Our construction relies on periodic randomized neighbor refreshing and pruning of high-degree nodes, and remains lightweight in terms of memory and messaging overhead. Our solution tackles one of the open problems in [5] by building directly on the framework developed by Augustine et al. [2].

The D-RAES Protocol. We now introduce D-RAES, a dynamic extension of the RAES protocol that is resilient to a continuous adversarial churn of up to $\mathcal{O}(n/\mathit{polylog}(n))$ nodes per round. The execution of the protocol is divided into two main phases: a bootstrap phase and

a maintenance phase. During the bootstrap phase, the goal is to construct a bounded-degree expander graph. In the maintenance phase, the protocol continually repairs and adapts the network in response to churn events. The bootstrap phase begins with a graph of n nodes and no edges. We run the RAES protocol (Algorithm 1) for $B = \mathcal{O}(\log n)$ rounds. As shown in [5], this results in an expander graph whp. (see Theorem 2.1 in [5]). After this phase, the adversary is allowed to churn nodes, and each node executes a maintenance cycle composed of three phases (Algorithm 2).

Algorithm 2 Overview of the D-RAES protocol.

Let G be the graph obtained after the Bootstrap Phase.
foreach $t > B$ **do**
  Phase 1 (refresh neighbors): With probability $1/\log^k n$ each node u with degree $d(u) \in [d, \Delta]$ drops all its neighbors.
  Phase 2 (reconnection): Each node u with degree $d(u) < d$, picks $d - d(u)$ new neighbors uniformly at random.
  Phase 3 (degree adjustment): Each node u with degree $d(u) > \Delta$ selects $d(u) - \Delta$ neighbors u uniformly at random and drops the connection with them.
end

The high level idea of the protocol is that each node in the network seeks to maintain a bounded degree graph by keeping a degree between the two parameters d and Δ with $d < \Delta$. This simple procedure mimics the behavior of the Bitcoin Network formation process [1]. We require that nodes in the network that were not churned by the adversary must refresh their neighbor list with probability $1/\mathit{polylog}(n)$. This is fundamental to avoid the oblivious adversary incrementally building bad structures that can gradually degrade and ultimately destroy the expansion of the graph. Thus, we require that on average, nodes must refresh their neighborhood every $\mathit{polylog}(n)$ rounds. Indeed, the authors in [7, 8] experimentally showed that refreshing long-lasting edges in the network might help the process to recover after a churn. The protocol uses a simple mechanism to understand when it is time to reconnect: when the number of neighbors falls below d it infers that is not well-connected anymore and it tries to find new random neighbors from the network. In addition, the protocol preserves bounded degrees by "pruning" neighbors of nodes that exceed Δ.

As mentioned above, our goal is to keep the network close to a random constant-degree graph at every round, w.h.p. Throughout, we fix the degree parameters d and Δ as constants with $d < \Delta$. Beyond the reconnection triggered by lost edges, in every round each node independently, with probability $\Theta(1/\mathit{polylog}(n))$, simply deletes all its edges and reconnects again. This refreshing step continuously creates new "random enough" edges in the network, which is useful in showing expansion properties in each round. Moreover, it is possible to show that if we do not include such a phase in the protocol (or an analogous step) it is impossible to maintain a constant degree expander graph under adversarial churn.

▶ **Theorem 1.** *Assume that after the bootstrap phase, the graph G_t is a constant-degree expander where every node has degree in $[d, \Delta]$, where $d < \Delta$ are suitable constants. Suppose that the maintenance protocol does not include any edge-refreshing phase. Then, there is an adversarial strategy that (1) maintains a constant degree graph in which each node has degree in $[d, \Delta]$ and (2) the graph is not anymore an expander graph.*

We show a high-level overview of the proof. The core idea is that there is an adversarial strategy that gradually erodes the well connected part of the graph while maintaining a bounded-degree graph. Without the edge-refresh phase (and thus using the RAES protocol

as it is in Algorithm 1) the adversary can: churn out some nodes from the graph and immediately insert $n/polylog(n)$ nodes by connecting these new nodes to a low-expansion region and avoid touching the core that keeps being an expander. However, over multiple rounds the well-connected part of the graph shrinks due to the churn while the weakly connected fringe grows. Since we do not have any mechanism to re-randomize the bad edges we have that the expansion collapses even though the graph is still a bounded-degree graph. Furthermore, we observe that a similar lower bound holds even when the adversary is oblivious, i.e., has no knowledge of the graph structure.

▶ **Observation 2.** *The lower bound in Theorem 1 holds under oblivious adversary.*

The graph produced after each iteration of Algorithm 2 (lines 2-6), is the result of the interaction between two entities: the oblivious adversary and Algorithm 2. To analyze the distributed protocol behavior under churn, we use the same approach of Augustine et al. [2] and we define a family of processes whose behavior is: (1) the adversary's move; (2) the actions by the nodes; and, (3) the random bits available to the process itself. As in [2], we refer to a member of this family of processes as the Υ-process.

Overview of the process. We present a high-level description of the process and intuitions behind its analysis and we refer to the full version of this work [6] for a complete description and analysis of the process. The main idea is that the process captures both, the distributed protocol's and the adversary's actions by first applying the adversarial churn to the graph and then steps (1-3) in Algorithm 2 at each round. We have that the Υ-process starts from a bounded degree expander graph that was created during the bootstrap phase. At each iteration it generates a new graph $G_t = (V_t, E_t)$, given the graph at $t-1$. The process ensures that nodes with a degree less than d and a life-span greater than $\Omega(\log n)$ will gain at least once a degree between d and Δ in $\mathcal{O}(\log n)$ rounds. Moreover, to argue that at each round we have a large expander subgraph we have to take into account that: (1) the (adversarial) churn is $\mathcal{O}(n/\log^k n)$, (2) the number of nodes that choose to refresh their neighborhood is $\mathcal{O}(n/\log^k n)$ whp., and (3) the number of "unlucky" nodes that after the reconnection phase had degree exactly d and that after the degree adjustment phase lost some neighbors (due to some highly connected node with degree greater than Δ dropping a random subset of their neighbors) is $\mathcal{O}(n/\log^k n)$ whp. By iteratively using the results by Bagchi et al. in [4], the fact that nodes sample from a uniform distribution over V_t and that no node can have degree less than d for $\Omega(\log n)$ consecutive rounds whp. we can argue that at the end of each iteration t, the Υ process produces a graph G_t that contains a large core with good expansion properties. This result implies that the D-RAES protocol maintains a graph with a $(n - o(n))$-sized constant expander subgraph in which all the nodes have degree between d and Δ. Our main result can be summarized as follows.

▶ **Theorem 3** (Main Theorem). *Let d and Δ be two suitable constants such that $d < \Delta$. Despite an adversarial churn rate of $\mathcal{O}(n/\log^k n)$ for any $k \geq 1$, the D-RAES protocol maintains a dynamic graph $\mathcal{G} = \{G_t = (V_t, E_t)\}_{t\geq 1}$ such that, with high probability, for at least n^c rounds (for any arbitrarily large constant $c \geq 1$), each snapshot G_t contains a connected subgraph C_t on $n - o(n)$ nodes that satisfies:*

- *each node in C_t has degree in $[d, \Delta]$;*
- *C_t has edge expansion of at least α, for some constant $\alpha > 0$.*

Our analysis and guarantees also hold under an adaptive adversary, provided that nodes always have access to uniform sampling over the current vertex set at each round. In this setting, although the adversary can observe the graph structure and churn history before

making decisions, it cannot predict or influence the outcome of random sampling. This implies that nodes can still establish fresh, well-distributed connections, and our edge-refreshing mechanism ensures that poorly connected regions do not persist. As a result, the same arguments used under the oblivious adversary extend to the adaptive case. However, we emphasize that this holds only under perfect uniform sampling. If uniform sampling is replaced with random walk-based sampling (as in [2]), an adaptive adversary can bias walk trajectories, making the analysis significantly more delicate and outside the scope of this work.

4 Conclusion

We presented a fully distributed and lightweight protocol for maintaining a constant-degree expander graph despite a continuous adversarial churn of up to $\mathcal{O}(n/polylog(n))$ nodes per round. Despite this strong adversary, our protocol ensures that the resulting overlay graph remains an expander with high probability over time. Moreover, this work demonstrates that robustness and good connectivity can be achieved under far weaker assumptions, without relying on global knowledge or heavy coordination among nodes. Our analysis suggests that, if nodes in the network periodically refresh their neighborhood, the Bitcoin Network creation Protocol can maintain a good expander graph that preserves a good edge expansion against heavy adversarial churn by simply executing a local dynamics in which nodes do not need any information about the overall network structure. Finally, our result closes an open problem in [5] where the authors suggested to study their protocol under churn settings.

References

1 Bitcoin core p2p network. URL: https://en.wikipedia.org/wiki/Bitcoin_Core.

2 John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.29.

3 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. SIAM, 2012. doi:10.1137/1.9781611973099.47.

4 Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. The effect of faults on network expansion. *Theory Comput. Syst.*, 2006. doi:10.1007/S00224-006-1349-0.

5 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Finding a bounded-degree expander inside a dense one. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. SIAM, 2020. doi:10.1137/1.9781611975994.80.

6 Antonio Cruciani. Maintaining a bounded degree expander in dynamic peer-to-peer networks, 2025. doi:10.48550/arXiv.2506.17757.

7 Antonio Cruciani and Francesco Pasquale. Brief announcement: Dynamic graph models for the bitcoin P2P network: Simulation analysis for expansion and flooding time. In *Stabilization, Safety, and Security of Distributed Systems - 24th International Symposium, SSS 2022, Clermont-Ferrand, France, November 15-17, 2022, Proceedings*, Lecture Notes in Computer Science. Springer, 2022. doi:10.1007/978-3-031-21017-4_23.

8 Antonio Cruciani and Francesco Pasquale. Dynamic graph models inspired by the bitcoin network-formation process. In *24th International Conference on Distributed Computing*

and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023. ACM, 2023. doi:10.1145/3571306.3571398.

9 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

Brief Announcement: Faster CONGEST Approximation Algorithms for Maximum Weighted Independent Set in Sparse Graphs

Salwa Faour
University of Freiburg, Germany

Fabian Kuhn
University of Freiburg, Germany

Abstract

The maximum independent set problem is a classic optimization problem in graph theory that has also been studied quite intensively in the distributed setting. Although the problem is hard to approximate within reasonable factors in general, there are good approximation algorithms known for several sparse graph families. In the present paper, we consider deterministic distributed CONGEST algorithms for the weighted version of the problem in trees and graphs of bounded arboricity (i.e., hereditary sparse graphs).

For trees, we prove that the task of deterministically computing a $(1-\varepsilon)$-approximate solution to the maximum weight independent set (MWIS) problem has a tight $\Theta(\log^*(n)/\varepsilon)$ complexity. The lower bound already holds on unweighted oriented paths. On the upper bound side, we show that the bound can be achieved even in unrooted trees.

For graphs $G=(V,E)$ of arboricity $\beta>1$, we give two algorithms. If the sum of all node weights is $w(V)$, we show that for any $\varepsilon>0$, an independent set of weight at least $(1-\varepsilon)\cdot\frac{w(V)}{4\beta}$ can be computed in $O(\log^2(\beta/\varepsilon)/\varepsilon+\log^* n)$ rounds. This result is obtained by a direct application of the local rounding framework of Faour, Ghaffari, Grunau, Kuhn, and Rozhoň [SODA '23]. We further show that for any $\varepsilon>0$, an independent set of weight at least $(1-\varepsilon)\cdot\frac{w(V)}{2\beta+1}$ can be computed in $O(\log^3(\beta)\cdot\log(1/\varepsilon)/\varepsilon^2\cdot\log n)$ rounds. For $\varepsilon=\omega(1/\sqrt{\beta})$, this significantly improves on a recent result of Gil [OPODIS '23], who showed that a $1/\lfloor(2+\varepsilon)\beta\rfloor$-approximation to the MWIS problem can be computed in $O(\beta/\varepsilon\cdot\log n)$ rounds. As an intermediate step to our result, we design an algorithm to compute an independent set of total weight at least $(1-\varepsilon)\cdot\sum_{v\in V}\frac{w(v)}{\deg(v)+1}$ in time $O(\log^3(\Delta)\cdot\log(1/\varepsilon)/\varepsilon+\log^* n)$, where Δ is the maximum degree of the graph.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases CONGEST model, weighted independent set, approximation, trees, arboricity

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.54

Related Version *Full Version*: https://arxiv.org/abs/2506.10845

1 Introduction and Related Work

Given a graph $G=(V,E)$ with node weights $w:V\to\mathbb{R}_{\geq 0}$, the *maximum weight independent set (MWIS)* problem asks for an independent set $I\subseteq V$ (i.e., a set I of pairwise non-adjacent nodes) such that the total weight $w(I):=\sum_{v\in V}w(v)$ is maximized. The MWIS problem is a classic optimization problem on graphs, which has also been studied quite extensively in the distributed setting. In the present paper, we focus on sparse families of graphs for which good MWIS approximations are known in the centralized and also in the distributed setting. We start by briefly summarizing the most relevant existing literature.

In the standard setting, the graph $G=(V,E)$ on which we intend to solve some given graph problem (e.g., MWIS) is also the communication graph. There are $n=|V|$ nodes and each node is equipped with an $O(\log n)$-bit unique identifier. The nodes V communicate over the edges in synchronous rounds. In the LOCAL model [18], the nodes can exchange


© Salwa Faour and Fabian Kuhn;
licensed under Creative Commons License CC-BY 4.0



arbitrarily large messages. In the more restricted CONGEST model, in each round, each node can send a (possibly different) message of $O(\log n)$ bits to each neighbor. When considering CONGEST algorithms for the MWIS problem, we further assume that a single node weight can be communicated with a single message. The internal computation at the nodes is not restricted. Initially, the nodes do not know anything about the topology of the graph G. For simplicity, we however assume that the nodes do know the values of the relevant parameters of G. At the end of an independent set algorithm, each node must know if it is in the independent set or not.

1.1 Distributed MWIS Algorithms in General Graphs

The most widely studied distributed independent set problem is the problem of computing a maximal independent set (MIS), i.e., an independent set that cannot be extended. The distributed complexity of computing an MIS has been studied intensively since the 1980s, e.g., [1, 5, 10, 9, 17, 11, 15]. The current best randomized algorithms have complexity $O(\log \Delta) + \tilde{O}(\log^{5/3} \log n)$ in LOCAL and $O(\log \Delta) + \tilde{O}(\log^3 \log n)$ in CONGEST [10, 9, 11].[1] The best known deterministic MIS algorithms require $\tilde{O}(\log^{5/3} n)$ rounds in LOCAL and $\tilde{O}(\log^2 \Delta \cdot \log n)$ rounds in CONGEST [9, 11]. For the *maximum cardinality independent set (MCIS)* problem, i.e., for the unweighted version of the MWIS problem, an MIS directly gives a $1/\Delta$-approximation (where Δ denotes the maximum degree of the graph). In [3], it is shown that in CONGEST, a $1/\Delta$-approximation for MWIS can be computed in time $O(\log W \cdot T_{\text{MIS}})$, where W denotes the ratio between the largest and smallest node weight and T_{MIS} is the time to compute an MIS. Subsequently, Kawarabayashi, Khoury, Schild, and Schwartzman [14] give an algorithm to compute an independent set of weight at least $(1-\varepsilon) \cdot \frac{w(V)}{\Delta+1}$ in time $O(1/\varepsilon)$ times the time to compute an independent set of weight at least $\frac{w(V)}{c(\Delta+1)}$ for some constant $c > 0$. For the latter problem, they give two algorithms. One of them uses the local-ratio technique [2] to reduce the problem to computing a single MIS of a (locally computable) subgraph of the input graph. The other one is a randomized algorithm that requires $\text{poly}(\log \log n)$ rounds. In [9], it is shown that it suffices to run $O(\log(1/\varepsilon))$ instances of computing an independent set of weight $\frac{w(V)}{c(\Delta+1)}$. Further, the paper gives a deterministic $O(\log^2 \Delta + \log^* n)$-time CONGEST algorithm to compute an independent set of total weight at least $\frac{w(V)}{4(\Delta+1)}$.

1.2 Distributed MWIS Algorithms for Sparse Graphs

We now get to the case of sparse graphs, which are the focus of this paper. It has been shown by Turán [19] that every graph of average degree $\overline{d}$ has an independent set of size at least $n/(\overline{d}+1)$. A natural generalization of this is the so-called Caro-Wei bound [7, 20], which states that every graph G has an independent set of size

$$\alpha(G) \geq \mathsf{CaroWei}(G) := \sum_{v \in V} \frac{1}{\deg(v)+1} \geq \frac{n}{\overline{d}+1}. \tag{1}$$

The second inequality follows from an application of the Cauchy-Schwarz inequality. As observed by Boppana, the Caro-Wei bound can be obtained in expectation by a simple random process that can be implemented in a single round in the CONGEST model [6, 13]. Every node picks an independent random number from a sufficiently large domain and a

[1] The notation $\tilde{O}(\cdot)$ hides polylogarithmic factors in the argument, i.e., $\tilde{O}(x) = x \cdot \text{poly} \log x$.

node joins the independent set if and only if it picked a smaller number than all its neighbors. This algorithm is equivalent to running a single phase of Luby's classic MIS algorithm [17]. In [6], it is shown that the Caro-Wei bound (1) is a $\frac{4\cdot(\sqrt{2}-1)}{\overline{d}+2} \approx \frac{1.657}{\overline{d}+2}$-approximation of the MCIS problem. For sparse graphs, we often want to express the approximation factor as a function of the arboricity β of the graph. The arboricity of a graph is defined as the number of forests into which the edges of a graph can be partitioned. Since we always have $\beta \geq 2\overline{d}$, the result of [6] implies that (1) is a $\frac{2\cdot(\sqrt{2}-1)}{\beta+1} \approx \frac{0.828}{\beta+1}$-approximation of the MCIS problem. In [9], it is shown that there is a deterministic CONGEST algorithm with round complexity $O\big(\frac{\log^2(\Delta/\varepsilon)}{\varepsilon} + \log^* n\big)$ that comes within a factor $(1/2-\varepsilon)$ of the bound of (1), as well as of a natural generalization of (1) in the weighted case. As one of the technical results of the present paper, we show that one can improve the $(1/2-\varepsilon)$ factor to a $(1-\varepsilon)$ factor at the cost of an additional $\log(\Delta)/\varepsilon$ factor in the round complexity (cf. Theorem 4).

The first paper that achieves an efficient $1/O(\beta)$-approximation for the MWIS problem in the CONGEST model is [14]. They achieve a $(1-\varepsilon)/(8\beta)$-approximation by first decomposing the graph into $O(\log n)$ layers of degree at most $4\beta-1$ and by running an algorithm to obtain an independent set of weight $(1-\varepsilon)\cdot w_i(V_i)/(4\beta)$ within each layer V_i (and where w_i is an appropriate weight function that is used in layer V_i). The overall round complexity is $O(\log n)$ times the time for computing the independent set of weight $(1-\varepsilon)\cdot w_i(V_i)/(4\beta)$ in each layer. By using the randomized algorithm of [14], the overall complexity of the resulting randomized $(1-\varepsilon)/(8\beta)$-approximation for the MWIS problem is $O\big(\frac{\log n\cdot \mathrm{poly}\log\log n}{\varepsilon}\big)$ and by using the deterministic algorithm of [9], the overall complexity of the resulting deterministic $(1-\varepsilon)/(8\beta)$-approximation for the MWIS problem is $O\big((\log^2\beta\cdot\log(1/\varepsilon)\cdot\log n\big)$. In fact, at the cost of an additional $1/\varepsilon$-factor in the round complexity, the approximation quality of the algorithm of [14] can be improved to $(1-\varepsilon)/(4\beta)$. Those results were then improved by Gil in [12], who in particular provides a deterministic CONGEST algorithm to compute a $1/\lfloor(2+\varepsilon)\beta\rfloor$-approximation in time $O(\beta\cdot\log(n)/\varepsilon)$.

More special classes of sparse graphs are trees and more generally minor-closes families of graphs (e.g., planar graphs). In [16, 8], it was shown that on cycles (and paths), deterministically computing a constant approximation for the maximum cardinality independent set problem requires $\Omega(\log^* n)$ rounds. Note that in all graphs of arboricity $\beta = O(1)$, computing a constant MCIS approximation is trivial. Half the nodes have degree at most 4β and an independent set of size $\Omega(n)$ among the nodes of degree at most 4β can be computed by computing an MIS in time $O(\beta+\log^* n)$ [5]. For planar graphs (and thus also for trees), it is shown in [8] that in the LOCAL model, even for the MWIS problem, a $(1-\varepsilon)$-approximation can be computed in time $\mathrm{poly}(\varepsilon^{-1})\cdot O(\log^* n)$.

1.3 Our Contributions

In our paper, we focus on deterministic approximation algorithms for the MWIS problem in the CONGEST model in sparse families of graphs. In the following, we list our technical contributions in detail and we also give an overview over the most important ideas that are needed to prove the stated theorems.

1.3.1 Approximating Maximum Weight Independent Set in Trees

We start by establishing a tight bound for computing a $(1-\varepsilon)$-approximation in trees.

▶ **Theorem 1.** *Let $\varepsilon > \log(n)/n$ be a parameter. The deterministic* CONGEST *model complexity of computing a $(1-\varepsilon)$-approximate solution for MWIS in tree networks is $\Theta\big(\frac{\log^* n}{\varepsilon}\big)$. The upper bound holds in general unrooted trees. The lower bound even holds for the unweighted version of the problem in oriented paths and in the* LOCAL *model.*

The lower bound is based on a relatively simple reduction from the MCIS problem in paths [8, 16]. The condition $\varepsilon > \log(n)/n$ is a technical condition because for arbitrary $\varepsilon > 0$, the lower bound becomes $\Omega(\log^*(\varepsilon n)/\varepsilon)$.

The upper bound is based on ideas that were developed for planar graphs in [8] and which are based the following core idea. First, one defines a weight function on edges, where for $\{u, v\} \in E$, we set $w(\{u, v\}) := \min\{w(u), w(v)\}$. The algorithm of [8] then computes a clustering of the nodes of V such that the total weight of the edges connecting nodes in different clusters is at most an $\varepsilon/2$-fraction of the total weight of all the edges. By computing an optimal weighted independent set within each cluster, one can then obtain a $(1-\varepsilon)$-approximate MWIS solution of the given instance. The independent sets of the clusters are combined by taking the union and removing the smaller weight node of every intercluster edge. The clustering algorithm of [8] obtains clusters of diameter $\text{poly}(\varepsilon^{-1})$ in time $\text{poly}(\varepsilon^{-1}) \cdot O(\log^* n)$ in the LOCAL model. In trees, the clustering algorithm of [8] can be implemented in the CONGEST model. Further, in trees, an optimal MWIS solution inside each cluster can be computed in time linear in cluster diameter by using a straightforward dynamic programming algorithm. The core challenge for proving the upper bound of Theorem 1 is to obtain a clustering algorithm in which the maximum cluster diameter is only $O(1/\varepsilon)$. The algorithm of [8] consists of basic steps in which the cluster diameter grows by constant factor and the total weight of the intercluster edges shrinks by a constant factor. The cluster diameter however grows by a factor that is larger than the factor by which the total weight of the intercluster edges decreases. In our algorithm, we show that in trees, one can interleave those basic steps of [8] with steps in which the clusters are split into clusters of smaller diameter, without increasing the overall weight of the intercluster edges by too much.

1.3.2 Approximation of MWIS as a Function of the Arboricity

For the remainder of this section, we assume that we are given a graph $G = (V, E)$ of arboricity β. In [14], it is shown that there is a deterministic CONGEST algorithm to compute an independent set of weight at least $\frac{w(V)}{(4+\varepsilon)\beta}$ in randomized time $O\big(\frac{\log n \cdot \text{poly}\log\log n}{\varepsilon}\big)$ and in combination with an algorithm from [9] in deterministic time $O\big(\frac{\log^2 \beta \cdot \log(1/\varepsilon) \cdot \log n}{\varepsilon}\big)$. In the following, we show that for moderately small β both those bounds can be improved significantly.

▶ **Theorem 2.** *For any $\varepsilon > 0$, there is a $O\big(\frac{\log^2(\beta/\varepsilon)}{\varepsilon} + \log^* n\big)$-round deterministic* CONGEST *algorithm to compute an independent set I of weight $w(I) \geq (1-\varepsilon) \cdot \frac{w(V)}{4\cdot\beta}$ in any graph $G = (V, E)$ of arboricity $\beta \geq 1$ and with node weights $w : V \to \mathbb{R}_{\geq 0}$.*

The theorem can be proven by a relatively straightforward blackbox application of the local rounding framework of [9]. More concretely, we can show that Theorem 2 follows almost directly from Lemma 4.2 in [9]. Theorem 2 implies a $\frac{1-\varepsilon}{4\beta}$-approximation of MWIS. In [12], it is shown that a $\frac{1}{\lfloor(2+\varepsilon)\beta\rfloor}$-approximation can be computed deterministically in $O\big(\frac{\beta\cdot\log n}{\varepsilon}\big)$ CONGEST rounds. For $\varepsilon = 1/o(\sqrt{\beta})$, we significantly improve on this bound as stated by the following theorem.

▶ **Theorem 3.** *For any $\varepsilon > 0$, there is a $O\big(\frac{\log^3(\beta)\cdot\log(1/\varepsilon)\cdot\log n}{\varepsilon^2}\big)$-round deterministic* CONGEST *algorithm to compute an independent set I of weight $w(I) \geq (1-\varepsilon) \cdot \frac{w(V)}{2\beta+1}$ in any graph $G = (V, E)$ of arboricity $\beta \geq 1$ and with node weights $w : V \to \mathbb{R}_{\geq 0}$.*

A major step towards proving Theorem 3 is an efficient algorithm to compute an independent set of weight arbitrarily close to a natural weighted generalization of the Caro-Wei bound (1). Specifically, we prove the following theorem.

▶ **Theorem 4.** *For any $\varepsilon > 0$, there is a $O\big(\frac{\log^3(\Delta)\cdot\log(1/\varepsilon)}{\varepsilon} + \log^* n\big)$-round deterministic* CONGEST *algorithm to compute an independent set I of weight $w(I) \geq (1-\varepsilon)\cdot\sum_{v\in V}\frac{w(v)}{\deg(v)+1}$ in any graph $G=(V,E)$ of maximum degree $\Delta \geq 1$ and with node weights $w : V \to \mathbb{R}_{\geq 0}$.*

Before discussing the main ideas needed to prove, Theorem 4, we discuss its direct application to approximating the maximum cardinality independent set problem in graphs of arboricity β. As proven in [6] and discussed in Section 1.2, the Caro-Wei bound (1) provides a $\frac{2\cdot(\sqrt{2}-1)}{\beta+1}$-approximation to the maximum cardinality independent set problem in graphs of arboricity β. Theorem 4 thus implies that such an approximation can be computed in $O\big(\frac{\log^3(\Delta)\cdot\log(1/\varepsilon)}{\varepsilon} + \log^* n\big)$ rounds in the CONGEST model. This can be further improved slightly. Because in a graph G of arboricity β, the average degree is at most 2β, the number of nodes of degree $> c\cdot\beta^2/\varepsilon$ is at most $2\varepsilon n/(c\beta)$, which can be shown to be at most $4\varepsilon/c\cdot\alpha(G)$, where $\alpha(G)$ denotes the independence number of G. It therefore suffices to apply Theorem 4 to the subgraph induced by the nodes of degree at most $c\beta^2/\varepsilon$ for a sufficiently large constant c. We thus obtain the following corollary.

▶ **Corollary 5.** *For any $\varepsilon > 0$, there is a $O\big(\frac{\log^3(\beta/\varepsilon)\cdot\log(1/\varepsilon)}{\varepsilon} + \log^* n\big)$-round deterministic* CONGEST *algorithm to compute a $(1-\varepsilon)\cdot\frac{2\cdot(\sqrt{2}-1)}{\beta+1}$-approximation to the maximum cardinality independent set problem in graphs of arboricity $\beta \geq 1$.*

We next sketch how Theorem 4 is proven and how it is used to prove Theorem 3. The starting point to proving Theorem 4 is an $O\big(\frac{\log^2\Delta\cdot\log(1/\varepsilon)}{\varepsilon} + \log^* n\big)$-round algorithm from [9] to compute an independent set of weight at least $(1-\varepsilon)\cdot\frac{w(V)}{\Delta+1}$ in a graph of maximum degree Δ. The idea now is to divide the nodes of G into $O(\log(\Delta)/\varepsilon)$ degree classes so that the degrees within one class differ by at most a factor $1+\varepsilon$. We then want to essentially apply the algorithm from [9] separately for each degree class, starting from the small degrees. This would work almost directly for unweighted graphs. To make it work, one just has to include all the still available nodes from the lower degree classes when computing the independent set of some degree class.

For weighted graphs, sequential composition of independent set is a bit more tricky. We can however utilize a technique developed in the context of the local-ratio method [2] and used in the context of distributed MWIS approximation in [3, 14, 9]. Given a graph G with node weights $w(v)$ and an independent set I_0, one can define a new weight function $w'(u) := \max\{0, w(u) - w(I_0 \cap N^+(u))\}$, where $N^+(u)$ denotes the inclusive neigborhood of u. If one then computes a second independent set I' consisting only of nodes v for which $w'(v) > 0$, the combined independent set $I := I' \cup (I_0 \setminus N^+(I'))$ is guaranteed to have a total weight of at least $w(I_0) + w'(I')$ (w.r.t. to the original weight function $w(\cdot)$). In this way, we can iterate through the $O(\log n/\varepsilon)$ degree classes in a similar way as in the unweighted case to obtain an independent set satisfying the requirements for Theorem 4.

We next discuss our algorithm to obtain the bound claimed by Theorem 3. Note that since the average degree of a graph of arboricity β can be almost 2β, we might have to compute an independent set of weight close to $w(V)$ divided by the average degree. There are graphs of arboricity β for which this is best possible (e.g., unweighted cliques) and we therefore in some cases cannot afford to allow too much "slack" when iteratively computing an independent set. A standard tool to algorithmically deal with graphs of bounded arboricity is the so-called H-decomposition as introduced in [4]. Because the average degree of a graph with arboricity

β is at most 2β, the number of nodes of degree more than $(2+\varepsilon)\beta$ is at most $(1-\Theta(\varepsilon))\cdot n$. When peeling off all nodes of degree at most $(2+\varepsilon)\beta$, we therefore get rid of a $\Theta(\varepsilon)$-fraction of all the nodes. Repeating this idea gives a decomposition of the graph into $O(\log(n)/\varepsilon)$ layers so that the nodes in each layer have at most $(2+\varepsilon)\beta$ neighbors in the same layer and higher layers. We are not aware of an algorithm to compute an independent set of weight close to $w(V)/(2\beta)$ that does not at least implicitly use an H-decomposition (which in most cases leads to a time complexity that is at least linear in the number of layers $\Theta(\log(n)/\varepsilon)$).

The highlevel idea of our approach is the following. We again use the framework of [2, 3, 14] to iteratively construct an independent set in the weighted setting. Because the nodes in the lowest layer are always the ones of bounded degree, we start our iterative construction at the lowest layer and work our way up the decomposition. Some of the main challenges with this approach already appear for the unweighted case and in the following highlevel discussion, we therefore focus on this case. Let V_0 be the nodes of the lowest layer of our decomposition. All nodes in V_0 have degree at most $(2+\varepsilon)\beta$ and we can therefore compute an independent set $I_0 \subseteq V_0$ of size close to $|V_0|/(2\beta)$ (or even close to the Caro-Wei bound (1) within V_0). However, even if the independent set I_0 is relatively large, it could consist of nodes that mostly have neighbors in $V \setminus V_0$ so that a lot of nodes in V_0 have no neighbor in I_0. When moving to the next layer, we can however not include any remaining nodes in V_0 because this might create some high degree nodes in the layer, we are considering. Our goal therefore should be to select an independent set I_0 in V_0 such that it removes most of the nodes in V_0. As an estimate of the number of removed nodes in V_0, we use $\sum_{v\in I_0}(\deg_0(v)+1)$, where $\deg_0(v)$ is the degree of v within V_0. If this sum is close to $|V_0|$, we are guaranteed to make sufficient progress (even if many nodes of V_0 remain because other nodes in V_0 have multiple neighbors in I_0). We can compute such an independent set I_0 as follows. We define a weight $\overline{w}(v) := \deg_0(v)+1$ for each node $v \in V_0$. Note that for an independent set I_0, we have $\overline{w}(I_0) = \sum_{v\in I_0}(\deg_0(v)+1)$. By using Theorem 4, we can compute an independent set I_0 of total weight $\overline{w}(I_0) \approx \sum_{v\in V_0} \frac{\overline{w}(v)}{\deg_0(v)+1} = |V_0|$, which is exactly what we need.

References

1 N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

2 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. of the ACM (JACM)*, 48:1069–1090, 2001. doi:10.1145/502102.502107.

3 Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *Proc. of 36th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 165–174, 2017. doi:10.1145/3087801.3087806.

4 L. Barenboim and M. Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using Nash-Williams decomposition. In *Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 25–34, 2008.

5 L. Barenboim, M. Elkin, and F. Kuhn. Distributed $(\Delta+1)$-coloring in linear (in Δ) time. *SIAM J. Computing*, 43(1):72–95, 2015.

6 Ravi B. Boppana, Magnús M. Halldórsson, and Dror Rawitz. Simple and local independent set approximation. In *Proc. 25th Int. Coll. on Structural Information and Communication Complexity (SIROCCO)*, pages 88–101, 2018. doi:10.1007/978-3-030-01325-7_12.

7 Y. Caro. New results on the independence number. Technical report, Tel Aviv University, 1979.

8 A. Czygrinow, M. Hańćkowiak, and W. Wawrzyniak. Fast distributed approximations in planar graphs. In *DISC*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-87779-0_6.

9 S. Faour, M. Ghaffari, C. Grunau, F. Kuhn, and V. Rozhon. Local distributed rounding: Generalized to mis, matching, set cover, and beyond. In *Proc. of the 2023 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2023), Florence, Italy, January 22-25, 2023*, pages 4409–4447, SIAM, 2023.

10 M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proc. 27th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 270–277, 2016.

11 Mohsen Ghaffari and Christoph Grunau. Near-optimal deterministic network decomposition and ruling set, and improved MIS. In *Proc. 65th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 2148–2179, 2024. doi:10.1109/FOCS61266.2024.00007.

12 Y. Gil. Improved Deterministic Distributed Maximum Weight Independent Set Approximation in Sparse Graphs. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, pages 16:1–16:20, 2024.

13 Magnús M. Halldórsson and Christian Konrad. Computing large independent sets in a single round. *Distributed Computing*, 31:69–82, 2018. doi:10.1007/S00446-017-0298-Y.

14 K. Kawarabayashi, S. Khoury, A. Schild, and G. Schwartzman. Improved distributed approximations for maximum independent set. In *Proc. 34th Symp. on Distributed Computing (DISC)*, pages 35:1–35:16, 2020.

15 F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *J. of the ACM*, 63(2), 2016. doi:10.1145/2742012.

16 C. Lenzen and R. Wattenhofer. Leveraging linial's locality limit. In *Proceedings of 22nd Symp. on Distributed Computing (DISC)*, pages 394–407, 2008.

17 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986. doi:10.1137/0215074.

18 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

19 P. Turán. On an extremal problem in graph theory (in Hungarian). *Mat. Fiz. Lapok*, 48:436–452, 1941.

20 V. K. Wei. A lower bound on the stability number of a simple graph. Technical Report No. 81-11217-9, Bell Laboratories Technical Memorandum, 1981.

Brief Announcement: Concurrent Double-Ended Priority Queues

Panagiota Fatourou
Department of Computer Science, University of Crete, Heraklion, Greece
FORTH ICS, Heraklion, Greece

Eric Ruppert
York University, Toronto, Canada

Ioannis Xiradakis
Department of Computer Science, University of Crete, Heraklion, Greece
FORTH ICS, Heraklion, Greece

Abstract

This work provides the first concurrent implementation of a double-ended priority queue (DEPQ). We describe a general way to add an ExtractMax operation to any concurrent priority queue that already supports Insert and ExtractMin.

2012 ACM Subject Classification Theory of computation → Concurrent algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases shared-memory, data structure, double-ended, priority queue, priority deque, heap, skip list, combining

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.55

Related Version *Full Version*: `https://arxiv.org/abs/2508.13399` [11]

Funding *Panagiota Fatourou and Ioannis Xiradakis:* Greek Ministry of Education, Religious Affairs and Sports through the HARSH project (project no. ΥΠ3ΤΑ - 0560901), within the framework of the National Recovery and Resilience Plan–Greece 2.0–with funding from the European Union–NextGenerationEU.
Eric Ruppert: Natural Sciences and Engineering Research Council of Canada.

1 Introduction

Priority queues, which store a set of keys and support an Insert operation that adds a key to the set and an ExtractMin operation that removes and returns the minimum key, have long been recognized as an important data structure for concurrent systems. They have been used in operating systems for job queues and load balancing, heuristic searches [29], graph algorithms (e.g., [1, 17]) and for event-driven simulations [19]. There are numerous concurrent implementations of priority queues.

In the single-process setting, there has been much research on designing a *double-ended* priority queue (DEPQ), which supports both ExtractMin and ExtractMax operations [31]. We provide a general transformation to construct a linearizable concurrent DEPQ from a concurrent (single-ended) priority queue. Implementing a *concurrent* DEPQ has not been explored previously.

We take our inspiration from the insight that some of the sequential DEPQ data structures can be viewed as being constructed from a pair of single-ended priority queues [5]. Our general construction in Section 3 uses two linearizable, concurrent priority queues to construct a linearizable *dual-consumer* DEPQ, which allows only one process to perform ExtractMin and one process to perform ExtractMax at a time. Insertions proceed concurrently with one another and with Extract operations. The construction works even if the underlying priority queues used are *single-consumer*, meaning that only one process at a time may perform ExtractMin operations. Our construction uses a new lightweight layer of synchronization between the two processes that perform ExtractMin and


© Panagiota Fatourou, Eric Ruppert, and Ioannis Xiradakis;
licensed under Creative Commons License CC-BY 4.0



ExtractMax operations. It preserves linearizability and lock-free progress: if the underlying priority queue is lock-free, then so is the resulting DEPQ. If the underlying priority queue also supports deletions of arbitrary elements, we provide time and space bounds for our DEPQ.

Section 4 describes how to adapt our dual-consumer DEPQ to handle concurrent Extract operations at each end using a lock-based combining technique [10]. At a high level, each process that acquires the lock for one end of the DEPQ performs a whole batch of Extract operations, which keeps the overhead for synchronization low.

Our technique is general enough to be applied to any concurrent priority queue. In the full version [11], we provide an example that applies our technique (including combining) to a simple list-based priority queue.

2 Related Work

There are many DEPQ implementations for just one process (see e.g., Chong and Sahni's survey [5]). In the concurrent setting, we are unaware of any previous work that aims to provide a linearizable concurrent DEPQ implementation. Medidi and Deo [24] described a DEPQ that supports batches of parallel operations, but their focus was on the synchronous PRAM model. Our construction uses two concurrent (single-ended) priority queues as building blocks. See [7, 26] for surveys of early work on concurrent priority queues, including many based on sequential heap data structures. Tamir, Morrison and Rinetzky [34] added support for an operation that modifies an existing key in a lock-based concurrent heap. Pugh's skip list data structure [28] has been used as the basis for several concurrent priority queues [3, 21, 32, 33]. Liu and Spear [22] gave a novel concurrent priority queue data structure based on a tree where each node stores a sorted linked list.

The concurrent priority queues discussed above allow multiple concurrent consumers. Our construction requires only a single-consumer priority queue, which may be easier to implement. Hoover and Wei [16] gave a wait-free single-consumer priority queue where operations take $O(\log n + \log p)$ steps, where n is the number of elements in the queue and p is the number of processes accessing it.

One way to use existing data structures to build a linearizable concurrent DEPQ is to use a binary search tree (BST), where the tree is sorted by key values [20, Section 5.2.3]. There are a number of lock-free concurrent BST implementations (e.g., [8, 27]) that support insertion and deletion of keys. The Delete operation can easily be modified to delete (and return) the minimum or maximum key present in the BST to yield the Extract operations of a DEPQ. Since repeated Extract operations at one end of the DEPQ could yield a lopsided BST, it would likely be desirable to use a balanced BST, such as the concurrent chromatic tree, which has both lock-based and lock-free implementations [2, 4]. However, even with balancing, this would require each Extract operation to traverse a path of length $\Theta(\log n)$ in the BST when the DEPQ contains n elements. In contrast, if we apply our approach to a (single-ended) concurrent priority queue based on a list or skip list, Extract operations find the required key right at the beginning of the list, and less restructuring is required, compared to the rebalancing of chromatic trees. There are also concurrent implementations of (single-ended) priority queues that augment a search tree with a sorted singly-linked list of keys in the tree to expedite ExtractMin operations [18, 30].

3 Constructing a Dual-Consumer DEPQ from Two Priority Queues

Our DEPQ construction uses two (single-ended) priority queues *MinPQ* and *MaxPQ* organized using opposite total orders on the keys. Thus, an Extract on *MinPQ* returns the minimum key and an Extract on *MaxPQ* returns the maximum key. Algorithm 1 gives pseudocode for our construction.

An **Insert** operation on the DEPQ simply inserts the key into both priority queues. To coordinate extractions, each item has an associated *reserved* bit, which is initially 0. An **ExtractMin** on the DEPQ repeatedly removes the (next) minimum element from *MinPQ* and tries to set the element's *reserved* bit using a TestAndSet instruction until it successfully changes the *reserved* bit of some element. That element is then returned. If, at any time, the ExtractMin observes that *MinPQ* is empty, it terminates and indicates that the DEPQ is empty. The **ExtractMax** operation on the DEPQ is symmetric to ExtractMin, using *MaxPQ* in place of *MinPQ*.

The *reserved* bit ensures that an element cannot be returned twice by both an ExtractMin and an ExtractMax. An element removed from the DEPQ by an ExtractMin operation may remain in *MaxPQ* for some time, but if it is eventually removed from *MaxPQ* by an ExtractMax operation, the ExtractMax will skip the item because its TestAndSet operation will fail to set the item's *reserved* bit.

If the priority queue implementation we are using also supports a Delete operation, then we can add the optional lines 15 and 25 to the Extract operations. The DEPQ is linearizable regardless of whether these lines are included or not. When an ExtractMin operation removes an item from *MinPQ*, line 15 removes it from *MaxPQ*. Whether the inclusion of these lines improves performance may depend upon the underlying priority queue: if deleting an arbitrary element is not much more costly than extracting the minimum, or if performance of the priority queues would degrade significantly due to the presence of obsolete items, then it may be worthwhile to include lines 15 and 25.

3.1 Linearizability and Progress

There are challenges in showing that the DEPQ is linearizable. Since *MinPQ* and *MaxPQ* are updated separately by Insert operations, their contents may not exactly match. Moreover, since the *reserved* bit is updated *after* removing the item from *MinPQ* or *MaxPQ*, the order in which items' bits are set may be different from the order in which they are removed from the priority queues.

Since we have assumed that the implementations of *MinPQ* and *MaxPQ* are linearizable, the composability property of linearizability [15] allows us to consider operations applied to each of these priority queues as atomic steps. For the sake of simplicity in our presentation, we assume that all keys inserted into the DEPQ are distinct. This allows us to talk about the Insert that inserted the key extracted by some ExtractMax or ExtractMin operation without ambiguity.

Fix an execution α. We now describe how to linearize operations in α. We linearize each ExtractMin and ExtractMax when it performs a successful TestAndSet (or when it sees that *MinPQ* or *MaxPQ* is empty, in the case of operations that return *nil*). An Insert(x) adds x to *MinPQ* and then to *MaxPQ*. By default, we linearize the Insert when it adds x to *MaxPQ*. However, if an ExtractMin removes x from *MinPQ* and returns it before x is added to *MaxPQ*, we must shift the linearization point of the Insert earlier.

We now describe this linearization more formally. Consider an ExtractMin or ExtractMax operation e. If e never performs an Extract at line 12 or 22, it is not linearized, since it never accesses shared memory. If e does perform an Extract at line 12 or 22, let $last_e$ be the last step where e does so, and let $result_e$ be the key returned by this last ExtractMin performed by e.

L1. Linearize an ExtractMin or ExtractMax operation e at $last_e$ if either $result_e = nil$ or e performs a successful TestAndSet at line 14 or 24.

L2. Linearize each Insert operation at the earlier of

- **a.** its insertion into *MaxPQ* at line 8, or
- **b.** immediately before the ExtractMin operation on DEPQ that returns its item.

If neither of these events occur, then the Insert is not linearized.

Algorithm 1 Generic construction of a dual-consumer DEPQ from two single-consumer priority queues.

```
1:  class item
2:      Key key
3:      boolean reserved
4:  end item

5:  function Insert(Key x)
6:      item i := new item with key = x, reserved = 0
7:      MinPQ.Insert(i)
8:      MaxPQ.Insert(i)
9:  end function

10: function ExtractMin : Key
11:     while true do                          ▷ repeatedly extract element from MinPQ
12:         item x := MinPQ.Extract
13:         if x = nil then return nil         ▷ DEPQ is empty
14:         else if TestAndSet(x.reserved) = 0 then   ▷ return when TestAndSet succeeds
15:             MaxPQ.Delete(x)                ▷ this line is optional
16:             return x.key
17:         end if
18:     end while
19: end function

20: function ExtractMax : Key
21:     while true do                          ▷ repeatedly extract element from MaxPQ
22:         item x := MaxPQ.Extract
23:         if x = nil then return nil         ▷ DEPQ is empty
24:         else if TestAndSet(x.reserved) = 0 then   ▷ return when TestAndSet succeeds
25:             MinPQ.Delete(x)                ▷ this line is optional
26:             return x.key
27:         end if
28:     end while
29: end function
```

In the full version [11], we prove every operation that terminates is assigned a linearization point, and that the linearization points of operations are within their execution intervals. Combining these claims with Lemma 1, which is the cornerstone of our proof, we get Theorem 2, proving linearizability.

▶ **Lemma 1.** *In the sequential execution defined by the linearization, each* ExtractMin *or* ExtractMax *operation e returns $result_e$.*

▶ **Theorem 2.** *If MinPQ and MaxPQ are linearizable (single-consumer) priority queues, then Algorithm 1 is a linearizable dual-consumer DEPQ.*

The Insert operation on the DEPQ is lock- or wait-free if insertions on the underlying priority queues are. The ExtractMin and ExtractMax operations on the DEPQ are lock-free if extractions (and deletions, if lines 15 and 25 are included) on the underlying priority queues are. This is because the TestAndSet on line 14 or 24 can fail only if some other ExtractMin or ExtractMax operation has successfully performed the TestAndSet, and that other operation is guaranteed to terminate.

ExtractMin and ExtractMax operations on the DEPQ are not wait-free, even if the underlying priority queue is wait-free because one such operation may repeatedly fail its TestAndSet in every iteration. Modifying the construction to make it wait-free would probably require some coordination between the two processes performing extractions at opposite ends of the DEPQ.

We provide amortized bounds on step complexity, provided that lines 15 and 25 are included. We say that the amortized step complexity is $O(T_i)$ for insertions and $O(T_e)$ for extractions if, for every (finite) execution in which m_i Inserts and m_e ExtractMin and ExtractMax operations are invoked, the total number of steps in the execution is $O(m_i \cdot T_i + m_e \cdot T_e)$. Let n be the maximum number of elements in the DEPQ at any time, if operations are performed sequentially in the order of the linearization. Let c be the maximum point contention, that is, the maximum number of operations that are active at any one time. Assume that the underlying priority queues have space complexity $S(n, c)$ and that the (amortized) step complexity for their Insert, Extract and Delete operations are $T'_i(n, c), T'_e(n, c)$ and $T'_d(n, c)$, respectively. The maximum number of elements that are ever in *MinPQ* or *MaxPQ* is $O(n+c)$. Thus, the amortized step complexity for each insertion and extraction operation on the DEPQ is $O(T'_i(n + c, c))$ and $O(T'_e(n + c, c) + T'_d(n + c, c))$, respectively.

If we have bounds on the *expected* step complexity of the underlying data structure (for example, if the data structure is randomized, like a skip list) then the bounds described above hold for the expected amortized step complexity of operations on the DEPQ. If T_i is a bound on the worst-case time per insertion into the underlying priority queue, rather than an amortized bound, then the bound for insertions on the DEPQ is also a worst-case bound.

4 Constructing a Multi-Consumer DEPQ from a Dual-Consumer DEPQ

The construction provided in Section 3 permits a single Extract operation at each end of the DEPQ at a time. If multiple processes wish to perform Extract operations, we could use two locks: one for each end. A process must acquire the lock for one end of the DEPQ before performing an Extract on that end. Insert operations can proceed regardless of the locks.

To reduce the overhead required by locking, we can use software combining, wherein the process that acquires the lock performs its own work as well as the work of other processes that failed to acquire the lock. Combining is particularly useful for data structures like stacks, queues, deques and priority queues, which have hot spots of contention (e.g., [10, 13, 14, 9]). Early versions of this approach [12, 35] used a tree to combine requests for work. Hendler et al. [14] developed a more efficient approach called flat combining.

We use the CC-Synch combining algorithm of Fatourou and Kallimanis [10] for our DEPQ. This combining algorithm has improved efficiency due to an integrated scheme for both locking the data structure and organizing the requests for operations on it. CC-Synch implements a combining-friendly variant of a queue lock [6, 23, 25]. The queue that implements the lock is also used to store the active operations in FIFO order. Processes use this lock to choose a *combiner* process, which is delegated to perform a batch of pending operations. After announcing its operation, each process p that is not chosen as the combiner simply waits (by performing local spinning) until a combiner informs p that its requested operation has been completed and provides the result of its operation.

We use two instances of CC-Synch, one for each end of the DEPQ. To perform an Extract operation on the DEPQ, a process adds its operation to the FIFO queue of the appropriate instance of CC-Synch. When CC-Synch chooses a combiner process to perform a batch of Extract operations from the FIFO queue, the combiner performs the Extracts one by one using Algorithm 1. An Insert simply runs Algorithm 1 directly, without using CC-Synch. CC-Synch ensures that only one process acts as a combiner at a time, satisfying Algorithm 1's requirement that there be only one process performing Extract operations at each end of the DEPQ. This approach improves locality in accessing the data structure, since the combiner process performs an entire batch of Extract operations. It also avoids having a hotspot of contention because multiple Extract operations would typically access the same part of the data structure. We remark that there is little contention on the *reserved* bits of items: only the two Extract operations that extract an item from *MinPQ* and *MaxPQ* can ever access its *reserved* field, and the *reserved* field is ignored by *Insert* operations.

References

1 D. P. Bertsekas, F. Guerriero, and R. Musmanno. Parallel asynchronous label-correcting methods for shortest paths. *Journal of Optimization Theory and Applications*, 88(2):297–320, February 1996. doi:10.1007/BF02192173.

2 Joan Boyar, Rolf Fagerberg, and Kim S. Larsen. Amortization results for chromatic search trees, with an application to priority queues. *Journal of Computer and System Sciences*, 55(3):504–521, 1997. doi:10.1006/jcss.1997.1511.

3 Anastasia Braginsky, Nachshon Cohen, and Erez Petrank. CBPQ: high performance lock-free priority queue. In *Proc. 22nd International Conference on Parallel and Distributed Computing*, volume 9833 of *LNCS*, pages 460–474. Springer, 2016. doi:10.1007/978-3-319-43659-3_34.

4 Trevor Brown, Faith Ellen, and Eric Ruppert. A general technique for non-blocking trees. In *Proc. 19th ACM Symposium on Principles and Practice of Parallel Programming*, pages 329–342, 2014. doi:10.1145/2555243.2555267.

5 Kyun-Rak Chong and Sartaj Sahni. Correspondence-based data structures for double-ended priority queues. *ACM J. Exp. Algorithmics*, 5, December 2000. doi:10.1145/351827.351828.

6 Travis S. Craig. Building FIFO and priority-queuing spin locks from atomic swap. Technical Report 93-02-02, Department of Computer Science and Engineering, University of Washington, 1993.

7 Kristijan Dragičević and Daniel Bauer. A survey of concurrent priority queue algorithms. In *Proc. 22nd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–6. IEEE, 2008. doi:10.1109/IPDPS.2008.4536331.

8 Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel. Non-blocking binary search trees. In *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2010. doi:10.1145/1835698.1835736.

9 P. Fatourou and P. Papadogiannakis. Double-ended queues based on software combining. Manuscript, 2025.

10 Panagiota Fatourou and Nikolaos D. Kallimanis. Revisiting the combining synchronization technique. In *Proc. 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 257–266, 2012. doi:10.1145/2145816.2145849.

11 Panagiota Fatourou, Eric Ruppert, and Ioannis Xiradakis. Concurrent double-ended priority queues. *CoRR*, abs/2508.13399, 2025. URL: https://arxiv.org/abs/2508.13399.

12 James R. Goodman, Mary K. Vernon, and Philip J. Woest. Efficient synchronization primitives for large-scale cache-coherent multiprocessors. In *Proc. 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 64–75, April 1989. doi:10.1145/70082.68188.

13 O. Grimes, A. Hassan, P. Fatourou, and R. Palmieri. PIPQ: A strict insert-optimized concurrent priority queue. In *Proc. 39th International Symposium on Principles of Distributed Computing*, 2025.

14 Danny Hendler, Itai Incze, Nir Shavit, and Moran Tzafrir. Flat combining and the synchronization-parallelism tradeoff. In *Proc. 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 355–364, 2010. doi:10.1145/1810479.1810540.

15 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, July 1990. doi:10.1145/78969.78972.

16 Kenneth D. Hoover and Yuanhao Wei. A fast single-extractor wait-free priority queue. Manuscript. A brief overview in the *Review of Undergraduate Computer Science* is available from https://rucs-uoft.github.io/theory-of-computation/a-fast-single-extractor-wait-free-priority-queue, 2017.

17 Qin Huang and W.E. Weihl. An evaluation of concurrent priority queue algorithms. In *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, pages 518–525, 1991. doi:10.1109/SPDP.1991.218255.

18 Theodore Johnson. A highly concurrent priority queue. *J. Parallel Distributed Comput.*, 22(2):367–373, 1994. doi:10.1006/JPDC.1994.1097.

19 Douglas W. Jones. Concurrent simulation: an alternative to distributed simulation. In *Proc. 18th Winter Simulation Conference*, pages 417–423. ACM, 1986. doi:10.1145/318242.318468.

20 Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley Longman, third edition, 1998.

21 Jonatan Lindén and Bengt Jonsson. A skiplist-based concurrent priority queue with minimal memory contention. In *Proc. 17th International Conference on Principles of Distributed Systems*, volume 8304 of *LNCS*, pages 206–220. Springer, 2013. doi:10.1007/978-3-319-03850-6_15.

22 Yujie Liu and Michael F. Spear. Mounds: Array-based concurrent priority queues. In *Proc. 41st International Conference on Parallel Processing*, pages 1–10. IEEE Computer Society, 2012. doi:10.1109/ICPP.2012.42.

23 Peter S. Magnusson, Anders Landin, and Erik Hagersten. Queue locks on cache coherent multiprocessors. In *Proc. 8th International Symposium on Parallel Processing*, pages 165–171, 1994. doi:10.1109/IPPS.1994.288305.

24 Muralidhar Medidi and Narsingh Deo. Parallel min-max-pair heap. In *Proc. 13th IEEE International Phoenix Conference on Computers and Communications*, pages 322–328, 1994. doi:10.1109/PCCC.1994.504133.

25 John M. Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst.*, 9(1):21–65, 1991. doi:10.1145/103727.103729.

26 Mark Moir and Nir Shavit. Concurrent data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 48.8.1. Chapman and Hall/CRC, second edition, 2018. doi:10.1201/9781315119335.

27 Aravind Natarajan, Arunmoezhi Ramachandran, and Neeraj Mittal. FEAST: a lightweight lock-free concurrent binary search tree. *ACM Transactions on Parallel Computing*, 7(2):1–64, May 2020. doi:10.1145/3391438.

28 William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of ACM*, 33(6):668–676, 1990. doi:10.1145/78973.78977.

29 V. Nageshwara Rao and Vipin Kumar. Concurrent access of priority queues. *IEEE Trans. Computers*, 37(12):1657–1665, 1988. doi:10.1109/12.9744.

30 Adones Rukundo and Philippas Tsigas. TSLQueue: An efficient lock-free design for priority queues. In *Proc. 27th International Conference on Parallel and Distributed Computing*, volume 12820 of *LNCS*, pages 385–401. Springer, 2021. doi:10.1007/978-3-030-85665-6_24.

31 Sartaj Sahni. Double-ended priority queues. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 8. Chapman and Hall/CRC, second edition, 2018. doi:10.1201/9781315119335.

32 Nir Shavit and Itay Lotan. Skiplist-based concurrent priority queues. In *Proc. 14th International Parallel & Distributed Processing Symposium*, pages 263–268. IEEE Computer Society, 2000. doi:10.1109/IPDPS.2000.845994.

33 Håkan Sundell and Philippas Tsigas. Fast and lock-free concurrent priority queues for multi-thread systems. *J. Parallel Distributed Comput.*, 65(5):609–627, 2005. doi:10.1016/J.JPDC.2004.12.005.

34 Orr Tamir, Adam Morrison, and Noam Rinetzky. A heap-based concurrent priority queue with mutable priorities for faster parallel algorithms. In *Proc. 19th International Conference on Principles of Distributed Systems*, volume 46 of *LIPIcs*, pages 15:1–15:16, 2015. doi:10.4230/LIPICS.OPODIS.2015.15.

35 Pen-Chung Yew, Nian-Feng Tzeng, and Duncan H. Lawrie. Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Transactions on Computers*, 36(4):388–395, 1987. doi:10.1109/TC.1987.1676921.

Brief Announcement: Universal Dancing by Luminous Robots Under Sequential Schedulers

Caterina Feletti
Department of Computer Science, Università degli Studi di Milano, Italy

Paola Flocchini
School of Electrical Engineering and Computer Science, University of Ottawa, Canada

Debasish Pattanayak
Department of Computer Science and Engineering, Indian Institute of Technology Indore, India

Giuseppe Prencipe
Department of Computer Science, Università di Pisa, Italy

Nicola Santoro
School of Computer Science, Carleton University, Ottawa, Canada

Abstract

The `Dancing` problem requires a swarm of n autonomous mobile robots to form a sequence of patterns, aka perform a *choreography*. Existing work has proven that some crucial restrictions on choreographies and initial configurations (e.g., on repetitions of patterns, periodicity, symmetries, contractions/expansions) must hold so that the `Dancing` problem can be solved under certain robot models. Here, we prove that these necessary constraints can be dropped by considering the $\mathcal{LUMI}$ model (i.e., where robots are endowed with a light whose color can be chosen from a constant-size palette) under the quite unexplored *sequential scheduler*. We formalize the class of `Universal Dancing` problems which require a swarm of n robots starting from any initial configuration to perform a (periodic or finite) sequence of arbitrary patterns, only provided that each pattern consists of n vertices (including multiplicities). However, we prove that, to be solvable under $\mathcal{LUMI}$, the length of the feasible choreographies is bounded by the compositions of n into the number of colors available to the robots. We provide an algorithm solving the `Universal Dancing` problem by exploiting the peculiar capability of sequential robots to implement a distributed counter mechanism. Even assuming non-rigid movements, our algorithm ensures spatial homogeneity of the performed choreography.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Luminous Robots, Sequence of Patterns, Pattern Formation, Sequential Scheduler

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.56

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2508.15484` [10]

Funding This work was partly supported by NSERC through the Discovery Grant program.

Acknowledgements This research was done while Caterina Feletti was visiting Profs. Flocchini and Santoro.

1 Introduction

The *Look-Compute-Move* (LCM) model is a theoretical model used to study distributed systems of mobile robots, aka swarms [12]. Generally, a swarm is modeled as a set of punctiform mobile robots $\mathcal{R} = \{r_1, \ldots, r_n\}$ acting on the Euclidean plane $\mathbb{R}^2$; they are assumed to be *anonymous* and *indistinguishable* (i.e., no internal nor external ids), *autonomous* (no central control), and *homogeneous* (they execute the same deterministic algorithm). Any


© Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro;
licensed under Creative Commons License CC-BY 4.0



robot, idle by default, is repeatedly activated by a scheduler. As soon as activated, the robot performs an LCM cycle: it takes the snapshot of the system (*Look*), it executes the given algorithm calculating a position (*Compute*), and it moves straight towards the computed position (*Move*). By repeating the LCM cycle infinitely, the swarm performs a given task (aka solves a problem). Different LCM (sub) models, usually referred to as X^S, have been considered to study the solvability of a problem: $X = \mathcal{OBLOT}$ represents the most limited model where robots are both oblivious and silent, while $X = \mathcal{LUMI}$ represents the model where robots are embedded with a $O(1)$-size light whose color is used both as an internal state and a communication means. Instead, S refers to the synchronization scheduler setting: `FSYNCH`, `SSYNCH`, and `ASYNCH` are the most considered ones, while the *sequential* setting (`SEQ`), which activates only one robot at each round assuming the fairness condition, has remained nearly unexplored until a few years ago [3, 7, 8, 11, 15].

Interest in `SEQ` has been driven by recent research on the well-known `Pattern Formation` problem [1, 2, 9, 13, 14, 17, 18, 19]. `Pattern Formation` for a given pattern Π (i.e., a multiset of positions in $\mathbb{R}^2$) requires the robots to rearrange themselves into a configuration similar to Π, and terminate. Indeed, the features of the model, particularly its X and S, characterize the class of patterns that a swarm can form [20]. A question arose: under which model assumptions can a swarm solve the `Universal Pattern Formation` (`UPF`) problem, namely form *any pattern starting from any initial configuration*? An answer is given in [11] which shows that, except for point formation, `UPF` is solvable under $\mathcal{OBLOT}^{\texttt{SEQ}}$ without any additional assumption: consequently, `UPF`, with point formation, is solvable under $\mathcal{LUMI}^{\texttt{SEQ}}$.

Recognizing the power of `SEQ`, the same interest has turned towards the quite unexplored `Dancing` problem, which asks a swarm to form a periodic or finite sequence of patterns, aka a *choreography* [4, 5, 6]. In [6] it was shown that under $\mathcal{OBLOT}^{\texttt{SSYNCH}}$ robots can perform choreographies with several constraints (on patterns and choreography periodicity), assuming chirality agreement among robots; in [5], it has shown that, under $\mathcal{LUMI}$ and with a global chirality, the feasible choreographies are subject to fewer constraints, even assuming `ASYNCH` and non-rigid movements. Now, the question is similar: under which model assumptions can a swarm solve the `Universal Dancing` (`UD`) problem, namely perform *a sequence of arbitrary patterns starting from any initial configuration*?

This paper answers this question. We provide an algorithm that solves `UD` under $\mathcal{LUMI}^{\texttt{SEQ}}$ and considering robots that are *completely disoriented* (i.e., no agreement on origin, unit distance, axes, chirality) and suffering from *variable disorientation* (i.e., their local coordinate system may change from one LCM cycle to the next one). Robots have *strong multiplicity detection*, i.e., they can detect the number of all the robots – and their colors – lying in the same point (aka forming a multiplicity). We assume *non-rigid* movements; thus, any robot may be stopped unpredictably after having traveled at least a non-null constant distance δ, unknown to robots. Despite non-rigidity, our solution guarantees an interesting *spatial property* which provides homogeneity to the whole choreography performance: the smallest circles enclosing the patterns of the choreography (except for patterns with only two or three points) formed by the swarm are concentric. We prove that, to be solvable under $\mathcal{LUMI}$, the length/period of the feasible choreographies of `UD` is bounded by the compositions of n into the number of colors available to the robots. Our contribution is summarized here:

▶ **Theorem 1.** *The `Universal Dancing` problem can be solved under $\mathcal{LUMI}^{\texttt{SEQ}}$ by a swarm of n disoriented robots with $k \geq 4$ colors, assuming that the choreographies have length/period $q \leq \binom{n+k-7}{k-4}$, even assuming non-rigid movements.*

2 Preliminaries

Patterns and choreographies. A pattern Π is a multiset of positions in $\mathbb{R}^2$, called *vertices*. We denote with $shape(\Pi)$ the set containing the unique positions in Π. We distinguish between four classes of patterns, namely `Point`, `TwoPoints`, `ThreePoints`, and `NPoints`, according to $|shape(\cdot)|$, which is respectively 1, 2, 3, and more than 3. We denote with $\mathscr{P}(\Pi)$ the set of patterns similar[1] to Π. A swarm *forms* Π if it stops in a configuration whose multiset of positions is in $\mathscr{P}(\Pi)$.

A *choreography* is a sequence of patterns $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$ where $\Pi_{i+1} \notin \mathscr{P}(\Pi_i)$ and with $x \in \{1, \infty\}$. If $\mathcal{S}$ is *periodic* (i.e., if $x = \infty$), then we assume that $\Pi_0 \notin \mathscr{P}(\Pi_{q-1})$ and that the sequence $\Pi_0, \dots, \Pi_{q-1}$ cannot be written as $(\Pi_0, \dots, \Pi_{h-1})^{\frac{q}{h}}$ for any $h < q$. We say that q is the *length* (*period*, resp.) of $\mathcal{S}$ if $x = 1$ ($x = \infty$, resp.). A swarm $\mathcal{R}$ *performs* $\mathcal{S}$ if, for any $j = 1, \dots, x$, there exists a series of increasing finite times $\{t_{i+(j-1)q}\}_{i=0,\dots,q-1,j=1,\dots,x}$ such that $\mathcal{R}$ forms Π_i at time $t_{i+(j-1)q}$. If $x = 1$, then $\mathcal{R}$ must remain still after time t_{q-1}.

The `Dancing` problems. `Dancing` refers to the generic problem of performing choreographies. Multiple constraints can be applied to the set of choreographies in order to make them performable (aka *feasible*). Different constraints define specific `Dancing` problems, each one formalized as the triple $\mathfrak{D} = \langle \Sigma(n), \mathcal{I}(n), \phi \rangle$, where:

- $\Sigma(n)$ is a set of patterns with n vertices;
- $\mathcal{I}(n)$ is a set of initial configurations for a swarm of n robots;
- ϕ is a predicate on choreographies (e.g., limiting their length, pattern repetitions, etc.).

We say that $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$ is a *feasible* choreography for $\mathfrak{D}$ if $\Pi_i \in \Sigma(n)$ for any $i \in [0, q-1]$ and $\mathcal{S}$ satisfies ϕ. For $\mathfrak{D}$ to be well-defined, each pattern in $\Sigma(n)$ must appear in at least one feasible choreography. An algorithm $\mathbb{A}$ solves $\mathfrak{D}$ if, for any swarm $\mathcal{R}$ starting from a configuration in $\mathcal{I}(n)$, it makes $\mathcal{R}$ perform any feasible choreography for $\mathfrak{D}$. A `Dancing` problem $\langle \Sigma(n), \mathcal{I}(n), \phi \rangle$ is said to be `Universal` if $\Sigma(n) = (\mathbb{R}^2)^n$ (i.e., any pattern with n vertices), $\mathcal{I}(n) = (\mathbb{R}^2 \times \{\mathsf{off}\})^n$ (i.e., set of all possible initial configurations where robots are off-colored), and ϕ can only bound the length/period of the choreographies. In this paper, we solve the `Universal Dancing` under a specific bound ϕ. We prove that if ϕ is a tautology $\top$, then the problem is unsolved under our model.

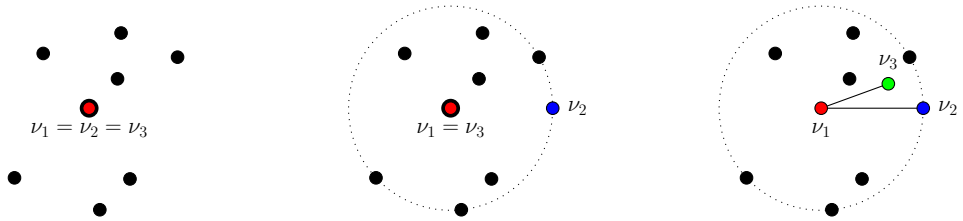
▶ **Theorem 2.** *The $\langle (\mathbb{R}^2)^n, (\mathbb{R}^2 \times \{\mathsf{off}\})^n, \top \rangle$ problem cannot be solved under $\mathcal{LUMI}^{SEQ}$ even if robots have rigid movements and share a global coordinate system.*

Proof. (*sketch*) A contradiction is achieved considering $\mathcal{S} = (\mathtt{P}_1, \Pi_1, \dots, \mathtt{P}_z, \Pi_z, \mathtt{P}_{z+1}, \Pi_{z+1})$, where $z = \binom{n+k-1}{k-1}$ is the number of color-compositions in a n-size swarm using k colors, and where $\mathtt{P}_i \in \mathtt{Point}$ for any $i \in [1, z+1]$ and $\Pi_i \neq \Pi_j$ for any $i \neq j \in [1, z+1]$. Then, there will be two times during the performance of $\mathcal{S}$ in which the swarm has the same configuration (same colors, forming $\mathtt{P}_a$ and $\mathtt{P}_b$, with $a < b$). This may lead the swarm to perform in a loop the sub-sequence $(\Pi_a, \dots, \Pi_{b-1}\mathtt{P}_b)^\infty$, thus never completing the choreography. ◀

3 Algorithm Outline

We here outline the algorithm that proves Theorem 1. Our algorithm uses a totally-ordered palette $\mathcal{L} = \{\ell_1, \dots, \ell_{k-3}, \mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3\}$ of $k \geq 4$ colors which allows a n-size swarm $\mathcal{R}$ to perform any feasible choreography $\mathcal{S} = (\Pi_0, \dots, \Pi_{q-1})^x$ with $x \in \{1, \infty\}$ that satisfies $q \leq \binom{n+k-7}{k-4}$. By convention, $\ell_1 = \mathsf{off}$, which is the default color to which all robots are initialized.

[1] We consider non-degenerate similarities.



(a) $\Pi_i \in$ `Point`. **(b)** $\Pi_i \in$ `TwoPoints`. **(c)** $\Pi_i \in$ `NPoints`.

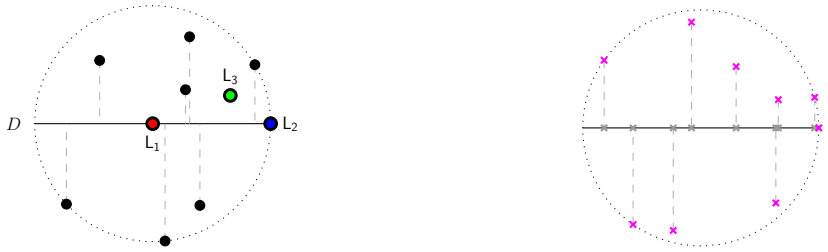
Figure 1 Chiral angles in different scenarios.

Techniques

Our algorithm adopts peculiar techniques which combine both the power of $\mathcal{LUMI}^{\texttt{SEQ}}$ and some concepts taken and adapted from other fields.

- **Universal orientation**. The `SEQ` setting allows us to elect three leaders, *starting from any configuration*. These leaders, colored with the colors L_1, L_2, and L_3, will be destined to form the *chiral angle*, a special triangle $\triangle\nu_1\nu_2\nu_3$ which globally defines a coordinate system for the swarm, and the position of the *actual pattern*, i.e., the pattern on whose vertices the robots must arrange. The construction of the chiral angle occurs before the formation of each pattern Π_i of the choreography and strictly depends on it; moreover, it is built to maintain spatial homogeneity to the choreography performance (e.g., ν_1 is always the same point for any Π_i). Notably, if $\Pi_i \notin$ `NPoints`, then the vertices of the chiral angle[2] exactly define the points where the robots have to arrange themselves. Instead, if $\Pi_i \in$ `NPoints`, ν_1 sets the origin $(0,0)$, ν_2 sets the coordinate $(0,1)$ thus the x-axis and the unit distance, and ν_3 sets the clockwise direction[3]. The actual pattern $\Pi \in \mathscr{P}(\Pi_i)$ is univocally defined by the chiral angle: ν_2 and ν_3 are two specific vertices of Π, and $SEC(\Pi)$ will be centered in ν_1, so that ν_2 lies on its boundary. See Figure 1.
- **Robot-vertex matching**. The formation of each pattern is based on a technique that makes the robots match their target vertices, exploiting properties of Dyck words on the Boolean alphabet.
- **Distributed counter**. We exploit the power of $\mathcal{LUMI}^{\texttt{SEQ}}$ by implementing a distributed *counter* to keep track of the pattern to form along the choreography. Specifically, the $n-3$ non-leader robots, using the colors $\ell_1, \ldots, \ell_{k-3}$, can encode an integer in the range $[0, z-1]$ where $z = \binom{n+k-7}{k-4}$ is the number of weak-compositions of $n-3$ robots with $k-3$ colors. For this purpose, we exploit the Gray Code for compositions in [16]: that method allows us to order all the composition vectors $X_0, \ldots, X_{z-1}$, so that each $X_i = (x_1, \ldots, x_{k-3})$ indicates the number of robots x_j of color ℓ_j, for any $j \in [1, k-3]$. The vector X_i thus encodes the counter value i. The Gray code ensures that X_0 corresponds to the vector where all the $n-3$ robots are `off`, and that X_i and X_{i+1} differ at exactly two indices, say a, b, so that $X_{i+1}[a] = X_i[a] + 1$ and $X_{i+1}[b] = X_i[b] - 1$: thus, to increment the counter, an ℓ_b-colored robot must turn into ℓ_a.

[2] Degenerate if $\Pi_i \in$ `Point` $\cup$ `TwoPoints`.
[3] When the vertices in Π_i are not aligned.



(a) Non-leader robots and their projections on D. **(b)** Projections of the target vertices, $\Pi_\perp$.

Figure 2 PHASE 2, case $\Pi \in$ `NPoints`.

Phases

The core of our algorithm is composed of three phases, i.e., PHASE 1, PHASE 2, PHASE 3, which are repeated cyclically until the choreography ends. For each $\Pi_i \in \mathcal{S}$, PHASE 1 aims at setting the chiral angle; PHASE 2 aims at making robots form Π_i; PHASE 3 aims at updating the counter value. Before entering the loop, robots execute an initial phase PHASE 0.

Phase 0 - Election of leaders. This phase is executed only once, starting from the initial configuration, and terminates when the three leaders have properly elected themselves by setting the dedicated colors L_1, L_2, and L_3. No robot moves. The counter value now is 0.

Phase 1 - Chiral angle setup. Let i be the counter's value, and let Π_i be the corresponding pattern in the choreography. In this phase, the three leaders L_1, L_2, L_3 compute the chiral angle $\triangle\nu_1\nu_2\nu_3$ for Π_i, and sequentially reach the relative vertices. No other robot moves.

Phase 2 - Pattern Formation. This phase starts when the leaders have formed the chiral angle for Π_i and ends as soon as each vertex of $\Pi \in \mathscr{P}(\Pi_i)$, unequivocally defined by the chiral angle, is covered by a robot. We describe PHASE 2 for the case $\Pi \in$ `NPoints`.

Let Ω be the circle centered in L_1 and such that L_2 lies on its boundary. By construction, all the robots are enclosed in Ω, and they will form Π so that $SEC(\Pi) = \Omega$. Note that L_2 and L_3 already lie on two vertices of Π, thus they will not move. Let D be the diameter of Ω starting from L_2. Given a vertex v of Π, we indicate with $v_\perp$ its projection on D. Robots arrange on Π in four sub-phases:

1. **Migration towards D.** Each non-leader robot reaches D moving along a perpendicular trajectory (see Figure 2a).
2. **Arrangement on D.** Once all $m = n - 3$ non-leader robots have reached D, they compute $\Pi_\perp = \{\!\{v_\perp \mid v \in \Pi\}\!\}$, i.e., the multiset of all the projections of Π (see Figure 2b), and remove from it the projections of the three vertices intended for the leaders. Let r be an activated robot on D. Thus, r computes the boolean string $w \in \{0,1\}^{\leq 2m}$ that represents the ordered arrangement of unmatched robots (1s) and uncovered projections (0s) along D. Note that multiplicities of robots (vertices, resp.) on D are treated by unrolling and representing them through factors of adjacent 1s (0s, resp.). Then, w can be factorized uniquely into Dyck words (or their reverses), which univocally defines the robot-projection matching (see Figure 3). Thus, each non-leader robot moves along D to reach the matching target projection. The properties of Dyck words ensure that each of the m non-leader robots will reach a vertex projection on D in finite time, even considering non-rigid movements.

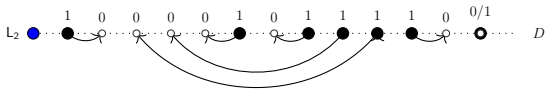


Figure 3 Robot-projection matching through Dyck words, where $w = 100001011110$.

3. **Back to the vertices.** In this sub-phase, all the non-leader robots travel perpendicularly to D and reach the corresponding vertices of Π. Let r be a non-leader robot on D: it unambiguously chooses the closest uncovered vertex of Π whose projection corresponds to its current position on D, and for which there does not yet exist a robot moving towards it. Note that multiple vertices (belonging to both the semi-circles cut by D) can have the same projection; in this case, r unambiguously elects the target vertex by considering the clockwise orientation of the plane. After the vertex choice, r starts traveling perpendicularly to D towards the computed vertex. Robot r may be stopped before reaching its vertex. Let h be the segment perpendicular to D which starts from D, contains r, and ends at Ω. Let $v_1, \ldots, v_s$ be the list of vertices along h so that $d(v_i, D) \geq d(v_{i+1}, D)$. Then r moves to v_j where $j \in [1, s]$ is the greatest index such that v_j is not covered and r lies on the segment $[v_j, v_{j_\perp})$.
4. L_1 **towards its vertex.** Lastly, L_1 moves to reach the last missing vertex of the pattern Π, which has been properly selected by the swarm.

Phase 3 - Counter Update. This phase starts as soon as each vertex of Π is covered by a robot. Let the current counter vector be $X_i = (x_1, \ldots, x_{k-3})$. We distinguish three actions:

- **Termination.** If $i = q - 1$ and $x = 1$, then the swarm has terminated the choreography, and then it does not update the counter: this causes the swarm to freeze in Π_{q-1}.
- **Increment.** If $i < q - 1$, then the swarm must increment the counter from i to $i + 1$. Let X_{i+1} be the next vector. From the Gray code construction, there are exactly two differences between X_i and X_{i+1} corresponding to one decrement value and one increment value. Let a and b be the two indices that change between X_i and X_{i+1}. Thus, as soon as a robot with color ℓ_a gets activated, it changes its color to ℓ_b.
- **Resetting.** If $i = q - 1$ and $x = \infty$, then the swarm must reset the counter to the initial value X_0, i.e., where all the non-leader robots are off-colored. For an unambiguous resetting, we make the leader L_3 set its color to L_2: the presence of two L_2 robots signals non-leader robots that they have to reset the counter. Once all non-leader robots have turned into off, then one of the two L_2 robots turns into L_3: if it is possible to establish which was the original L_3, the leaders maintain their former roles.

References

1 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Asynchronous arbitrary pattern formation: The effects of a rigorous approach. *Distributed Computing*, 32(2):91–132, 2019. doi:10.1007/S00446-018-0325-7.

2 Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Solving the pattern formation by mobile robots with chirality. *IEEE Access*, 9:88177–88204, 2021. doi:10.1109/ACCESS.2021.3089081.

3 Stefano Clemente and Caterina Feletti. Fault detection and identification by autonomous mobile robots. In *Proc. of 4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND)*, pages 10:1–10:20, 2025. doi:10.4230/LIPIcs.SAND.2025.10.

4 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Synchronized dancing of oblivious chameleons. In *Proc. of 7th International Conference on Fun with Algorithms (FUN)*, volume 8496, pages 113–124. Springer, 2014. doi:10.1007/978-3-319-07890-8_10.

5 Shantanu Das, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Forming sequences of patterns with luminous robots. *IEEE Access*, 8:90577–90597, 2020. doi:10.1109/ACCESS.2020.2994052.

6 Shantanu Das, Paola Flocchini, Nicola Santoro, and Masafumi Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, 2015. doi:10.1007/s00446-014-0220-9.

7 Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *Proc. of 20th International Symposium on Distributed Computing (DISC)*, volume 4167, pages 46–60. Springer, 2006. doi:10.1007/11864219_4.

8 Xavier Défago, Maria Potop-Butucaru, and Sébastien Tixeuil. *Fault-Tolerant Mobile Robots*. Chapter 10 of *[12]*, pages 234–251, 2019. doi:10.1007/978-3-030-11072-7_10.

9 Yoann Dieudonné, Franck Petit, and Vincent Villain. Leader election problem versus pattern formation problem. In *Proc. of 24th International Symposium on Distributed Computing (DISC)*, pages 267–281, 2010. doi:10.1007/978-3-642-15763-9_26.

10 Caterina Feletti, Paola Flocchini, Debasish Pattanayak, Giuseppe Prencipe, and Nicola Santoro. Universal dancing by luminous robots under sequential schedulers, 2025. arXiv:2508.15484.

11 Paola Flocchini, Alfredo Navarra, Debasish Pattanayak, Francesco Piselli, and Nicola Santoro. Oblivious robots under sequential schedulers: Universal pattern formation. In *Proc. 32nd International Colloquium of Structural Information and Communication Complexity (SIROCCO)*, volume 15671, pages 297–314. Springer, 2025. doi:10.1007/978-3-031-91736-3_18.

12 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors. *Distributed Computing by Mobile Entities: Current Research in Moving and Computing*, volume 11340. Springer, 2019. doi:10.1007/978-3-030-11072-7.

13 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Proc. of 10th International Symposium on Algorithms and Computation (ISAAC)*, pages 93–102, 1999. doi:10.1007/3-540-46632-0_10.

14 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1):412–447, 2008. doi:10.1016/j.tcs.2008.07.026.

15 Fabian Frei and Koichi Wada. Brief announcement: Distinct gathering under round robin. In *Proc. of 38th International Symposium on Distributed Computing (DISC)*, volume 319, pages 48:1–48:8, 2024. doi:10.4230/LIPIcs.DISC.2024.48.

16 Paul Klingsberg. A Gray code for compositions. *Journal of Algorithms*, 3(1):41–44, 1982. doi:10.1016/0196-6774(82)90006-2.

17 Giuseppe Prencipe. Pattern formation. In *Chapter 3 of [12]*, volume 11340, pages 37–62. Springer, 2019. doi:10.1007/978-3-030-11072-7_3.

18 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

19 Ramachandran Vaidyanathan, Gokarna Sharma, and Jerry Trahan. On fast pattern formation by autonomous robots. *Information and Computation*, 285:104699, 2022. doi:10.1016/j.ic.2021.104699.

20 Masafumi Yamashita and Ichiro Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26-28):2433–2453, 2010. doi:10.1016/j.tcs.2010.01.037.

Brief Announcement: The Virtue of Self-Consistency

Fabian Frei
CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Koichi Wada
Hosei University, Koganei, Tokyo, Japan

Abstract

We show that self-consistency can be a crucial property for autonomous mobile robots.

Specifically, we consider the task of gathering three robots, placed adversarially in distinct locations in the Euclidean plane, in a single point. We assume the natural scheduler RoundRobin, which activates the robots in turns. An activated robot perceives all robot locations in an adversarially scaled, rotated, and mirrored Cartesian coordinate system with itself at the origin and then moves wherever it wants. We show that this task cannot be solved in the default robot model (without any consistency guarantees and no multiplicity detection) but becomes feasible if we assume self-consistency (i.e., no changes between the different activations of the same robot) of either the unit length (i.e., no scaling) or the compass (i.e., no rotating) by providing explicit algorithms.

2012 ACM Subject Classification Computer systems organization → Robotic autonomy

Keywords and phrases Autonomous Mobile Robots, Distinct Gathering, Round Robin, Disorientation, Self-Consistency

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.57

Funding *Fabian Frei*: This research was funded in part by grant number 218001 awarded to Fabian Frei by the Swiss National Science Foundation.
Koichi Wada: Supported by JSPS KAKENHI No. 25K03079.

Acknowledgements We thank the reviewers for their helpful comments and suggestions. The work by Fabian Frei was done in part while at CISPA Helmholtz Center for Information Security, Saarbrücken.

1 Model and Motivation

The computational power of autonomously acting, simple, mobile robots has been the object of intense research in the field of distributed computing. Ever since Suzuki and Yamashita's seminal work [13], a large amount of work has been dedicated to the research of theoretical models of such autonomous mobile robots [1, 2, 3, 6, 10, 11, 12]. The standard textbook by Flocchini et al. [7] provides an excellent overview of the basic models and literature. In the standard model, each robot is represented as a point in the two-dimensional plane with limited capabilities. In particular, the robots are assumed to be *oblivious* (have no persistent memory), *anonymous* (have no IDs), *uniform* (run identical algorithms) [7]. This basic model is often called Oblot, which is an abbreviation of "*obl*ivious rob*ot*."

The robots operate in *Look-Compute-Move* (LCM) cycles. A robot performs such a cycle whenever it is activated by the so-called scheduler. For the scheduler RoundRobin, which we consider here, the robots are activated in turns, keeping the same, previously unknown, order forever. Whenever a robot is activated, it performs a full LCM cycle before the next robot is activated. Each cycle is composed of the following three successive operations: In the *Look* operation, a robot obtains a snapshot of the plane showing the positions of the other robots without multiplicity in a Cartesian coordinate system with the

© Fabian Frei and Koichi Wada;
licensed under Creative Commons License CC-BY 4.0

observing robot at the origin. In the *Compute* operation, it then executes its algorithm, which is identical for all robots, using the snapshot as input. Finally, it moves towards the computed destination in the *Move* operation.

We assume the robots to be *transparent*, i.e., not obstructing each other's views, and *collision-resistant*, i.e., passing each other without consequence.

By default, the choice of the Cartesian coordinate system is fully volatile; that is, any robot's coordinate system may be rotated, scaled, and mirrored adversarially before each activation. If the adversary cannot scale and rotate, we speak of a *consistent unit length* and *consistent compass*, respectively. If the adversary can scale and rotate between different robots but not between different activations of the same robot, we say that the unit length and compass are *self-consistent*, respectively.

We also distinguish between the standard model of *rigid movement* and the weaker assumption of *nonrigid movement*, which models various adverse situations such as robots overheating or running out of energy during their movements. The standard assumption for mobile robots executing Look-Compute-Move cycles is that they always reach their target. This is called *rigid movement*, as opposed to *nonrigid movement*, where a robot's movement may be stopped at any time by the adversary, as long as the robot has moved some minimum distance δ during its current cycle. The minimum movement distance δ is *unknown* to the robots. Note that the knowledge of a common δ would indeed provide the robots with a consistent unit length.

We consider the fundamental and extensively studied task of gathering all robots in a single, arbitrary point. Together with our assumption that the robots start from distinct locations, this task is also called *Distinct* Gathering, in contrast to *Self-Stabilizing* Gathering without this assumption. A recent survey summarizes results for both Distinct Gathering and Self-Stabilizing Gathering in the Euclidean plane [9].

It has been a widespread and natural assumption of the field that the purely robot-internal property of self-consistency cannot help them coordinate. We prove that this is not the case by showing that Distinct Gathering of three robots under RoundRobin is an unsolvable problem in the default model without consistency guarantees but feasible as soon as we assume either a self-consistent unit or a self-consistent compass. This shows that it is important to be careful and precise in model descriptions when it comes to assumptions about self-consistency or the lack thereof.

2 Related Work and Contribution

A quarter century ago, Suzuki and Yamashita [13] introduced in their seminal paper the robot model that served as the basis for much of the research on mobile robots during the following years. The key difference to what we present here is their assumption that robots can detect how many robots occupy any location. They show how gathering can be achieved under this assumption [13, Thm. 3.4]. The main idea is to use the geometry of the initial configuration to create an arbitrary point of multiplicity, and then let all robots move to the point of highest multiplicity. But depending on the given sensors, the robots might not be able to detect multiplicity. Moreover, multiplicity makes it rather simple to achieve a gathering, such that most problems in this settings were resolved quickly. It has thus become the standard assumption of the subsequent papers that robots cannot detect multiplicity. As previously stated, this is also our assumption here.

The remaining question is therefore whether the task remains feasible without multiplicity detection. While gathering $n = 2$ robots is trivial under RoundRobin, it remained a prominent open question whether Distinct Gathering is still possible for $n \geq 3$ robots. This

question was first officially raised at DISC 2006 [4], tackled repeatedly [5, 14], and finally resolved last year, as announced at DISC 2024 [8]: The problem is indeed infeasible for $n = 3$ robots, which matches the established conjecture by Défago et al. [5, Conj. 2], but surprisingly becomes solvable again for $n \geq 4$ robots.

For the purposes of this paper, however, the most interesting result is indeed the infeasibility of Distinct Gathering under RoundRobin for $n = 3$ robots in the default model without any additional assumptions introduced above.

The last and best attempt to find an algorithm solving the problem by introducing a minimal set of assumptions is due to Terai et al. [14, Sect. 4, Alg. 10]. They managed to find an algorithm for the problem if the algorithm is equipped with some internal memory (referred to as internal lights) and at the same time a consistent unit distance. The results announced in this paper show that the aforementioned result still made several unnecessary assumptions. Specifically, we can show that solving the problem requires neither internal nor external memory nor a consistent unit distance.

For the first of our two results, which is formally stated in Theorem 1, we provide an algorithm (Algorithm 1) that works with only the additional assumption that the robots have a self-consistent unit length. In particular, each robot may be adversarially assigned its own unit distance for any given initial configuration. As mentioned above, it is quite surprising that a purely self-consistent property can help the robots coordinate among each other at all. The conventional assumption has been that some shared knowledge or perception is required for the robots to improve their capabilities.

Motivated by the first result, we have also investigated the case where not the unit length, but instead a sense of direction, which we call a compass (essentially a vector of variable length always pointing in the same direction), is given to the robots in a self-consistent way. As it turns out, the problem is even solvable under this assumption, without any self-consistency for the unit length. This is stated in Theorem 2. Due to the space constraints, we cannot provide the full algorithm in this case and can only sketch the main idea. The techniques for the two proofs of Theorems 1 and 2 are quite dissimilar. Indeed, it seems impossible to simulate either of the two algorithms using the other one.

▶ **Theorem 1.** *Distinct Gathering under* RoundRobin *is possible for* 3 *robots with a self-consistent unit length and the otherwise default assumptions detailed in the introduction, even if movement may be nonrigid.*

Proof Sketch. The algorithm (Algorithm 1) relies on a distance classification scheme. Specifically, the continuum of positive real numbers $\mathbb{R}_{>0}$ is partitioned into two disjoint infinite unions of intervals:

- $\mathbb{D}_{\text{golden}} = \bigcup_{k \in 2\mathbb{Z}} [\varphi^{k+1}, \varphi^k)$,
- $\mathbb{D}_{\text{gather}} = \bigcup_{k+1 \in 2\mathbb{Z}} [\varphi^{k+1}, \varphi^k)$,

where $\varphi = \frac{2}{1+\sqrt{5}} = 0.618\ldots$ is the golden ratio. These intervals are logarithmically spaced and serve to discretize distance observations in a self-consistent manner.

The main challenge addressed by the algorithm is indeed the lack of consistency in perceived distances: each robot uses its own arbitrary unit scale. Hence, we cannot directly rely on any single robot's perception to define globally consistent behavior. The key idea is to enforce a structure on the observed distances that causes predictable transitions between configurations.

The algorithm operates on configurations of three robots, moving through the following high-level phases:

1. **Initialization and Contraction:** Starting from any arbitrary configuration (collinear or non-collinear), the robots execute moves that eventually reduce the diameter of the configuration without ever increasing it. Through repeated activations and based on the

classification of the observed triangle's side lengths, the algorithm gradually transforms any initial triangle into one with a unique shortest side. This prepares the configuration for controlled symmetry breaking.

2. **Controlled Symmetry Breaking:** Once a triangle with a unique shortest side is achieved, the robot not adjacent to this side is eventually activated and may attempt to move toward the other robots. However, this action is taken only when the perceived length of the shortest side falls into $\mathbb{D}_{\text{golden}}$. This restriction ensures that the two-point configuration created is well-structured and predictable in terms of how the observing robot will behave.
3. **Two-Point to Three-Point Cycling:** When a two-point configuration is created, the robot that initiated the move is now collocated with one of the other robots, while the third robot is at a different location. The key insight is that based on the perceived distance (again classified into $\mathbb{D}_{\text{golden}}$ or $\mathbb{D}_{\text{gather}}$), robots either remain stationary or move in small, controlled steps to create specific collinear three-point configurations. This cycling between two-point and three-point formations allows the robots to coordinate indirectly through geometry and ensures that enough variations of distance configurations are explored.
4. **Eventual Rigid Movement:** Another critical component of the algorithm is that it ensures a bounded decrease of the configuration diameter after a finite number of steps. Because the system allows only non-increasing diameter movements, and the distance thresholds used in the classification scheme are strictly decreasing with powers of φ, the configuration's diameter eventually becomes smaller than any possible non-rigid movement threshold. This guarantees that we can thereafter assume all robot moves to be rigid.
5. **Convergence to Gathering:** After sufficiently many transitions between specific two-point and three-point configurations, the robots arrive at a configuration where all three perceive distances from $\mathbb{D}_{\text{gather}}$ and are activated in one of a bounded number of scenarios. Each of these is explicitly handled in the algorithm and shown to lead to gathering within a finite number of steps, regardless of the scheduler's choices or initial scale mismatches. ◀

▶ **Theorem 2.** *Distinct Gathering under* RoundRobin *is possible for* 3 *robots with a self-consistent compass and the otherwise default assumptions detailed in the introduction, even if movement may be nonrigid.*

Proof Sketch. The algorithm, which we have to defer to the future full paper due to the space constraint, solves the Distinct Gathering problem under RoundRobin for three robots with self-consistent compasses, even when faced with nonrigid movement.

The overall strategy of the algorithm is structured in two main phases. The first phase achieves a collinear configuration in which all robots possess a well-defined and consistent sense of direction along the shared line. A robot derives such a sense of direction from its self-consistent compass vector if this vector is not orthogonal to the line formed by the robot positions. We can prove that such a collinear configuration with a self-consistent sense of direction for all robots eventually emerges from any initial configuration. Furthermore, we can show that the diameter of the configuration decreases over time as long as nonrigid movements occur, thus guaranteeing that rigid movement can eventually be assumed without loss of generality.

Once a collinear configuration as described is reached, the second phase of the algorithm begins. This phase relies on the robots' ability to break symmetry using their self-consistent compasses. The robots restrict their movements to the line formed by them and behave

Algorithm 1 An algorithm for Distinct Gathering under RoundRobin for $n = 3$ robots that works with potentially inconsistent but always self-consistent perception of unit length and for nonrigid movement with an unknown minimum movement distance for each robot and otherwise the default assumptions detailed in the introduction. We denote by φ the golden ratio $0.618\ldots$ solving the equation $\varphi + \varphi^2 = 1$. Moreover, we denote by $\mathbb{D}_{\text{golden}} = \cup_{k\in 2\mathbb{Z}}[\varphi^{k+1}, \varphi^k)$ and $\mathbb{D}_{\text{gather}} = \cup_{k+1\in 2\mathbb{Z}}[\varphi^{k+1}, \varphi^k)$ the two sets of the bipartition $(\mathbb{D}_{\text{golden}}, \mathbb{D}_{\text{gather}})$ of the positive real numbers.

Input: A distinct configuration of robots in the Euclidean plane, observed in a Cartesian coordinate system with the observing robot at the origin.

Output: A target location that the observing robot moves towards on a straight line, not arriving at the target only when interrupted by nonrigid movement.

Algorithm:

(1) **If** a *one-point configuration* (i.e., a gathering) is observed, **then** no movement is made.

(2) **If** a *two-point configuration* is observed, **then** there are two possible behaviors and which one is chosen depends on the perceived distance d between the two locations:
- **A.** **If** $d \in \mathbb{D}_{\text{gather}}$, **then** the opposite location is targeted.
- **B.** **If** $d \in \mathbb{D}_{\text{golden}}$, **then** the activated robot targets the point between the two locations that is at perceived distance $\varphi^2 d$ from the observation point and thus at perceived distance φd from the opposite location.

(3) **If** a *collinear three-point configuration* is observed, **then** the behavior depends on the observed relative distances:
- **A.** **If** the ratio between the two distances from the middle robot to the outer robots is the golden ratio and the observing robot is not the middle one, the behavior depends again on the observed distance d between the observing robot and the middle robot:
  - **a.** **If** $d \in \mathbb{D}_{\text{golden}}$, **then** the robot stays where it is.
  - **b.** **If** $d \in \mathbb{D}_{\text{gather}}$, **then** the behavior depends on whether the middle robot is closer to the observing robot or the other one.
    - α. **If** the middle robot is closer to the non-observing robot, **then** the target is the middle robot.
    - β. **If** the middle robot is closer to the observing robot, **then** the trajectory goes past the middle robot to the location between the middle robot and the other non-observing robot that was from the middle robot a distance that is a factor φ of the distance to the other non-observing robot.
- **B.** **If** the observing robot is the middle one, the behavior depends on the ratio between the two distances from the middle robot to the outer robots:
  - **a.** **If** the ratio is φ or $1/\varphi$, **then** the target is the second location with the ratio φ or $1/\varphi$ between the outer robots.
  - **b.** **If** the ratio is neither φ nor $1/\varphi$, **then** the activated robot moves towards an arbitrary one of the two points forming an isosceles right triangle with the outer two robots.

(4) **If** a robot observes a *non-collinear three-point configuration*, **then** the behavior depends on whether the observing robot is incident to any of the shortest sides in the triangle formed by the three robot locations:
- **a.** **If** the observing robot is incident to such a shortest side, **then** it targets the midpoint of this shortest side, breaking potential ties arbitrarily.
- **b.** **If** the observing robot is not incident to such a shortest side, **then** there is only one shortest side and the behavior depends on the observed length d of this side:
  - α. **If** $d \in \mathbb{D}_{\text{golden}}$, **then** the target is any arbitrary one of the two endpoints of the unique shortest side.
  - β. **If** $d \in \mathbb{D}_{\text{gather}}$, **then** the robot stays where it is.

differently depending on their observed relative positions and compass directions. The correctness of this phase is verified through an exhaustive case analysis of all 48 possible directional configurations that they eventually lead to a gathering.

Together, these two phases ensure that the algorithm terminates with all robots gathered at a single location. ◀

3 Conclusion

We have explored the minimal assumptions required to make the Distinct Gathering of three robots under ROUNDROBIN feasible. The most recent algorithm attempting to minimize the assumptions still required rigidity of movement, a common unit distance, and additionally memory. We have provided an algorithm that reduces all of this to only a self-consistent unit length. We further have an algorithm that requires only a self-consistent compass instead of the self-consistent unit. Both results highlight the importance of the commonly neglected distinction between robot perceptions that are potentially not consistent but at least self-consistent and fully volatile perceptions without any such guarantees.

References

1 N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006. doi:10.1137/050645221.

2 Z. Bouzid, S. Das, and S. Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *the 33rd Int. Conf. on Distributed Computing Systems*, pages 334–346, 2013.

3 M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012. doi:10.1137/100796534.

4 Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In Shlomi Dolev, editor, *Proceedings of the 20th International Symposium on Distributed Computing (DISC 2006)*, volume 4167 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2006. doi:10.1007/11864219_4.

5 Xavier Défago, Maria Potop-Butucaru, and Philippe Raipin Parvédy. Self-stabilizing gathering of mobile robots under crash or Byzantine faults. *Distributed Comput.*, 33(5):393–421, 2020. doi:10.1007/s00446-019-00359-x.

6 B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer. A tight run-time bound for synchronous gathering of autonomous robots with limited visibility. In *23rd ACM SPAA*, pages 139–148, 2011.

7 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan and Claypool Publishers, 2012.

8 Fabian Frei and Koichi Wada. Brief announcement: Distinct Gathering under Round Robin. *Leibniz International Proceedings in Informatics, LIPIcs*, 319, 2024. doi:10.4230/LIPICS.DISC.2024.48.

9 Fabian Frei and Koichi Wada. Invited paper: Gathering oblivious robots in the plane. In Toshimitsu Masuzawa, Yoshiaki Katayama, Hirotsugu Kakugawa, Junya Nakamura, and Yonghwan Kim, editors, *Stabilization, Safety, and Security of Distributed Systems – 26th International Symposium, SSS 2024, Nagoya, Japan, October 20–22, 2024, Proceedings*, volume 14931 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2024. doi:10.1007/978-3-031-74498-3_3.

10 T. Izumi, Z. Bouzid, S. Tixeuil, and K. Wada. Brief announcement: The BG-simulation for Byzantine mobile robots. In *25th DISC*, pages 330–331, 2011.

11 S. Kamei, A. Lamani, F. Ooshita, and S. Tixeuil. Asynchronous mobile robot gathering from symmetric configurations without global multiplicity detection. In *18th SIROCCO*, pages 150–161, 2011.

12 S. Souissi, X. Défago, and M. Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1):1–27, 2009. doi:10.1145/1462187.1462196.

13 Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999. doi:10.1137/S009753979628292X.

14 Satoshi Terai, Koichi Wada, and Yoshiaki Katayama. Gathering problems for autonomous mobile robots with lights. *Theor. Comput. Sci.*, 941:241–261, 2023. doi:10.1016/j.tcs.2022.11.018.

Brief Announcement: Optimal-Length Labeling Schemes for Fast Deterministic Communication in Radio Networks

Adam Ganczorz
Institute of Computer Science, University of Wrocław, Poland

Tomasz Jurdzinski
Institute of Computer Science, University of Wrocław, Poland

Andrzej Pelc
Département d'informatique, Université du Québec en Outaouais, Gatineau, Canada

Abstract

We consider two fundamental communication tasks in arbitrary radio networks: broadcasting (information from one source has to reach all nodes) and gossiping (every node has a message and all messages have to reach all nodes). Nodes are assigned labels that are (not necessarily different) binary strings. Each node knows its own label and can use it as a parameter in the same deterministic algorithm. The length of a labeling scheme is the largest length of a label. The goal is to find labeling schemes of asymptotically optimal length for the above tasks, and to design fast deterministic distributed algorithms for each of them, using labels of optimal length.

Our main result concerns broadcasting. We show the existence of a labeling scheme of constant length that supports broadcasting in time $O(D + \log^2 n)$, where D is the diameter of the network and n is the number of nodes. This broadcasting time is an improvement over the best currently known $O(D \log n + \log^2 n)$ time of broadcasting with constant-length labels, due to Ellen and Gilbert (SPAA 2020). It also matches the optimal broadcasting time in radio networks of known topology. Hence, we show that appropriately chosen node labels of constant length permit to achieve, in a distributed way, the optimal centralized broadcasting time. This is, perhaps, the most surprising finding of this paper. We are able to obtain our result thanks to a novel methodological tool of propagating information in radio networks, that we call a 2-height respecting tree.

Next, we apply our broadcasting algorithm to solve the gossiping problem. We get a gossiping algorithm working in time $O(D + \Delta \log n + \log^2 n)$, using a labeling scheme of optimal length $O(\log \Delta)$, where Δ is the maximum degree. Our time is the same as the best known gossiping time in radio networks of known topology.

2012 ACM Subject Classification Networks → Network algorithms; Computing methodologies → Distributed algorithms; Theory of computation → Distributed algorithms

Keywords and phrases radio network, distributed algorithms, algorithms with advice, labeling scheme, broadcasting, gossiping

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.58

Related Version *Full Version*: `https://arxiv.org/abs/2410.07382`

Funding Supported by the National Science Center, Poland (NCN), grant 2020/39/B/ST6/03288. *Andrzej Pelc*: Partially supported by NSERC discovery grant RGPIN 2024-03767 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

We consider two fundamental communication tasks often occurring in networks. In *broadcasting*, one node, called the *source*, has a message that must reach all other nodes. In *gossiping*, every node has a message and all messages have to reach all nodes.


© Adam Ganczorz, Tomasz Jurdzinski, and Andrzej Pelc;
licensed under Creative Commons License CC-BY 4.0



The model and the problem

We consider radio networks modeled as simple undirected connected graphs. Throughout this paper, $G = (V, E)$ denotes the graph modeling the network, n denotes the number of its nodes, D its diameter, and Δ its maximum degree. At the cost of a small abuse of notation, we sometimes use D to denote the height of a BFS spanning tree of a graph with a fixed root node. Note however that the height of a BFS tree is not larger than the diameter D and not smaller than $D/2$, so the orders of magnitude are the same. In our probabilistic considerations concerning graphs with n nodes, we use the term "with high probability" to mean "with probability at least $1 - 1/n$".

We use square brackets to indicate sets of consecutive integers: $[i, j] = \{i, \ldots, j\}$ and $[i] = [1, i]$. All logarithms are to the base 2. For simplicity of presentation, we assume throughout the paper that the number of nodes of a graph n is a power of 2, in order to avoid rounding of logarithms. One can easily generalize all the results for arbitrary n, preserving asymptotic efficiency measures.

As usually assumed in the algorithmic literature on radio networks, nodes communicate in synchronous rounds (also called steps). All nodes start executing an algorithm in the same round. In each round, a node can either transmit a message to all its neighbors, or stay silent and listen. At the receiving end, a node v hears the message from a neighbor w in a given round, if v listens in this round, and if w is its only neighbor that transmits in this round. If more than one neighbor of a node v transmits in a given round, there is a *collision* at v. Two scenarios concerning collisions were considered in the literature. The availability of *collision detection* means that node v can distinguish collision from silence which occurs when no neighbor transmits. If collision detection is not available, node v does not hear anything in case of a collision (except the background noise that it also hears when no neighbor transmits). We do not assume collision detection. The time of a deterministic algorithm for a given task is the worst-case number of rounds it takes to solve it, expressed as a function of various network parameters.

If nodes are indistinguishable (anonymous), i.e., in the absence of any labels, none of our communication problems can be solved, for example, in the four-cycle. Hence we consider labeled networks, i.e., we assign binary strings, called *labels*, to nodes. A *labeling scheme* for a given network represented by a graph $G = (V, E)$ is any function $\mathcal{L}$ from the set V of nodes to the set S of finite binary strings. The string $\mathcal{L}(v)$ is called the label of the node v. Labels assigned by a labeling scheme are not necessarily distinct. The *length* of a labeling scheme $\mathcal{L}$ is the maximum length of any label assigned by it. Every node knows a priori only its label, and can use it as a parameter in the same deterministic algorithm

Solving distributed network problems with short labels can be seen in the framework of algorithms with *advice*. In this paradigm that has recently got growing attention, an oracle knowing the network gives advice to nodes not knowing it, in the form of binary strings, provided to nodes before the beginning of a computation. A distributed algorithm uses this advice to solve the problem. The required size of advice (maximum length of the strings) can be considered a measure of the difficulty of the problem. Two variations are studied in the literature: either the binary string given to nodes is the same for all of them [16] or different strings may be given to different nodes [8, 7, 9, 10], as in the case of the present paper. If strings may be different, they can be considered as labels assigned to nodes by a labeling scheme. Such labeling schemes permitting to solve a given network task efficiently are also called *informative labeling schemes*. One of the famous examples of using informative labeling schemes is to answer adjacency queries in graphs [1].

Several authors have studied the minimum amount of advice (i.e., label length) required to solve certain problems (see the subsection Related work). The framework of advice permits us to quantify the minimum amount of information used to solve a given network problem, regardless of the type of information that is provided. Note that the scenario of the same advice given to all (otherwise anonymous) nodes would be useless in the case of radio networks: no deterministic communication could occur.

We now define formally our two communication tasks in a radio network $G = (V, E)$.

• **Broadcasting**: One node of the graph, called the *source*, has a broadcast message that has to reach all nodes $v \in V$. A node which already knows the broadcast message is called an *informed* node, otherwise the node is *uninformed*. If a node v receives the broadcast message for the first time in round r, from some neighbor u, we say that u *informed* v in round r. An uninformed node v is a *frontier node* in a given round, if it is a neighbor of an informed node. In our broadcasting algorithms, only informed nodes send messages.

• **Gossiping:** Each $v \in V$ has a message, and all messages have to reach all nodes in V.

As it is customary in algorithmic literature concerning radio networks, we assume that when a node sends a message, this message can be of arbitrary size. In particular, a node could send its entire history (however, in our algorithms, messages will be usually shorter: in broadcasting, some control messages will be appended to the source message, and in gossiping, all messages already known to a node will be combined in a single message).
Now our goal can be succinctly formulated as follows:

> For the above tasks, find an optimal-length labeling scheme permitting to accomplish this task, and design an optimal-time algorithm using a scheme of optimal length.[1]

Our results

Our main result concerns broadcasting. We improve the best currently known time of deterministic broadcasting using labeling schemes of constant length, due to Ellen and Gilbert (SPAA 2020) [7]. As in [7], our results are of two types: *constructive*, where the labeling scheme used by the algorithm is explicitly constructed using an algorithm polynomial in n, and *non-constructive*, where we only prove the existence of the labeling scheme used by the algorithm, via the probabilistic method. The broadcasting algorithm from [7] using a constructive constant-length labeling scheme runs in time $O(D \log^2 n)$. We improve it to time $O\left(D + \min(D, \log n) \cdot \log^2 n\right)$. The broadcasting algorithm from [7] using a non-constructive constant-length labeling scheme runs in time $O(D \log n + \log^2 n)$. We improve it to time $O(D + \log^2 n)$. This latter time is, in fact, the optimal deterministic broadcasting time in radio networks of known topology.[2]. Hence, we show that appropriately chosen node labels of constant length permit us to achieve, in a deterministic distributed way, the optimal centralized broadcasting time. This is, perhaps, the most surprising finding of this paper. We are able to obtain our result thanks to a novel tool that we call a 2-height respecting tree. The properties of these crucial objects permit us to implement centralized communication techniques in the distributed context.

It should be mentioned that messages used by both our broadcasting algorithms contain the source message and a constant number of control bits.

[1] For the task of broadcasting, constant-length labeling schemes are known, so in this case the goal is to find a scheme of constant length supporting an optimal-time broadcasting algorithm.

[2] This means that every node has an isomorphic copy of the graph, with nodes labeled in the same way by unique identifiers, and a node knows its identifier. Deterministic algorithms using such knowledge are called *centralized*.

Next, we apply our broadcasting algorithm to solve the gossiping problem. Using the non-constructive version of our result for broadcasting, we get an algorithm working in time $O(D + \Delta \log n + \log^2 n)$, that uses a (non-constructive) labeling scheme of optimal length $O(\log \Delta)$.[3] Our time is the same as the best *known* gossiping time for radio networks of known topology (without any extra assumptions on parameters), that follows from [13].

Table 1 Previous and our results.

Ref.	Time	Length of labeling scheme	Constructive
Broadcasting: centralized optimal time $O(D + \log^2 n)$, [13, 19]			
[8]	$O(n)$	2 bits	Yes
[7]	$O(D \log n + \log^2 n)$	3 bits	No
[7]	$O(D \log^2 n)$	3 bits	Yes
here	$O(D + \log^2 n)$	7 bits	No
here	$O\left(D + \min(D, \log n) \cdot \log^2 n\right)$	7 bits	Yes
Gossiping: centralized best time known $O(D + \Delta \log n + \log^2 n)$, follows from [13]			
here	$O(D + \Delta \log n + \log^2 n)$	$\Theta(\log \Delta)$	No
here	$O(D + \Delta \log n + \min(D, \log n) \log^2 n)$	$\Theta(\log \Delta)$	Yes

Related work

Broadcasting and gossiping in radio networks were extensively investigated in algorithmic literature. For deterministic algorithms, two important scenarios were studied. The first concerns centralized algorithms, in which each node knows the topology of the network and its location in it. Here, an optimal-time broadcasting algorithm was given in [13, 19] and the best known gossiping time (without any extra assumptions on parameters) follows from [13]. For large values of Δ, this was later improved in [4]. The second scenario concerns distributed algorithms, where nodes have distinct labels, and every node knows its own label and an upper bound on the size of the network but does not know its topology. Here the best known broadcasting time that depends only on n is $O(n \log n \log \log n)$ [20], later improved in [5] for some values of parameters D and Δ. For gossiping, the best known time in arbitrary directed (strongly connected) graphs was given in [12, 14] and the best known time for undirected graphs follows from [22]. Randomized distributed broadcasting was studied in [18, 6], where optimal $O(D \log(n/D) + \log^2 n)$ time algorithms were obtained independently. For gossiping, optimal randomized time was given in [15].

The advice paradigm has been applied to many different distributed network tasks: finding a minimum spanning tree [9], finding the topology of the network [10], and leader election [16]. In [8] and [7], the task was broadcasting in radio networks, as in the present paper. In the above papers, advice was given to nodes of the network. Other authors considered the framework of advice for tasks executed by mobile agents navigating in networks, such as exploration [17] or rendezvous [21]. In this case, advice is given to mobile agents.

[3] Using only constructive labeling schemes, the polylogarithmic summand in our complexity of gossiping changes from $\log^2 n$ to $\min(D, \log n) \log^2 n$.

2 High-level Description of our Results

High-level description of broadcasting

Our algorithms for broadcasting combine three mechanisms. The first two of them are taken from the literature, and the third, which is our main technical contribution, permits us to improve the time complexity of broadcasting.

1. The domination mechanism from [8]. Computation is split into blocks of some constant number of rounds. At the beginning of block r, a fixed set $\mathbf{DOM}_r$ of nodes is active which is a minimal set of informed nodes with respect to inclusion that covers all frontier nodes. All elements of $\mathbf{DOM}_r$ simultaneously transmit in the first round of the block called the *Broadcast* step. Minimality of $\mathbf{DOM}_r$ guarantees that each $v \in \mathbf{DOM}_r$ informs at least one uninformed node. For each $v \in \mathbf{DOM}_r$, the labeling algorithm chooses exactly one such node v' informed by v in block r as the feedback node of v in that block.

All feedback nodes can transmit simultaneously messages received by the nodes which serve as their witnesses. These feedback nodes transmit in the second round of the block, called the *Feedback* step. Their messages contain some information stored in their labels which instruct the corresponding nodes from $\mathbf{DOM}_r$ whether they should stay in $\mathbf{DOM}_{r+1}$ and instruct them about their behaviour in the remaining steps of the current block r.

Nodes informed until block r which are outside of $\mathbf{DOM}_r$ remain inactive to the end of an execution of the algorithm. The intuition regarding this property is the fact that a node v outside of $\mathbf{DOM}_r$ does not have its feedback node to instruct v about its actions. On the other hand, v cannot store this information in its own label for many blocks of computation, because it would require non-constant size of labels.

As each block extends the set of informed nodes, we have broadcasting in $O(n)$ time.

2. The propagation mechanism from [7]. In order to accelerate propagation of the broadcast message in the case when the diameter D of the input graph is $o(n)$, ideas from a randomized seminal distributed algorithm of Bar-Yehuda et al. [2] are applied. Namely, for appropriate random choices of informed nodes whether to transmit in a particular round, one can assure that the broadcast message is passed to the consecutive level of a BFS tree rooted at the source node s in $O(\log n)$ rounds in expectation. This in turn gives randomized broadcasting in $O(D \log n + \log^2 n)$ rounds.

These random choices of nodes are mimicked in the labels of nodes. More precisely, the labels store some 0/1 random choices whether to transmit in a given block, assuring a given time bound. In particular, the feedback node of a node $v \in \mathbf{DOM}_r$ stores, in the bit *Go* of its label, information whether v should transmit. Then, the nodes from $\mathbf{DOM}_r$ which received *Go*=1 transmit the broadcast message in the separate *Go* step of the block r. The labeling scheme obtained in this way is non-constructive. Using ideas from [3] regarding centralized broadcasting in arbitrary bipartite graphs, one can obtain a constructive labeling scheme. However, the time of the broadcasting algorithm such a scheme would support becomes $O(D \log^2 n)$ instead of $O(D \log n + \log^2 n)$ supported by the non-constructive scheme.

3. The fast tracks mechanism. This mechanism is the main novelty of our solution and permits us to improve the broadcasting time from [7]. The goal here is to implement ideas of a fast *centralized* algorithm into constant-size advice such that a *distributed* algorithm can somehow simulate the centralized one. The key ingredient of our approach is illustrated by the notion of a 2-height respecting trees (2-HRT) and the fact that there exists a BFS

tree which is also 2-HRT, for each graph. The 2-height of a node v in a tree intuitively denotes the maximum number of "critical branches" (causing large congestion) on a path from v to a leaf. The maximum 2-height is always at most $\log n$. Each time the 2-height of a node v and of some child w of v are the same, transmission of a message from v to w can be made in parallel with other similar transmissions from the level of v dedicated to the particular value of 2-height. Therefore, such an edge connecting v and w with equal 2-heights is called a *fast edge*. As all but $\log n$ edges on each path from the root to a leaf are fast, the centralized algorithm from [13] accomplishes broadcast in almost optimal time $O(D + \log^3 n)$. With this aim, the authors of [13] make use of the notion of gathering trees which somehow minimize collisions between fast edges. Our notion of a 2-HRT imposes stronger requirements than gathering trees, making fast transmissions even more parallelizable. Then, the key challenge is an implementation of the idea of a centralized algorithm by constant-size labels instructing nodes of a distributed algorithm how to simulate the centralized algorithm. The main obstacle here comes from the domination mechanism which switches off some nodes irreversibly, preventing them from transmitting any message starting from the block r in which they are outside of the minimal dominating set $\mathbf{DOM}_r$. We show that, for each such node, one can determine its "rescue node" still present in the dominating set, such that its transmission on behalf of a switched off node does not cause additional collisions.

Our final solution using this mechanism gives a non-constructive labeling scheme of constant length, supporting broadcasting in time $O(D + \log^2 n)$, which is optimal, even for centralized algorithms. Using the technique from [3] we can build labels constructively at the cost of increasing the time complexity of broadcasting to $O\left(D + \min(D, \log n) \cdot \log^2 n\right)$.

High-level description of gossiping

We introduce the auxiliary task of *gathering*: each node of the graph has a message, and all messages have to reach a designated node called the *sink*. We provide a gathering algorithm working in time $O(D + \Delta \log n + \log^2 n)$ and using a labeling scheme of length $O(\log \Delta)$.

With this aim, we make use of properties of a 2-HRT to implement the centralized algorithm for gathering from [13] in a distributed way, using short labels. Let T be a BFS tree of the input graph which is also a 2-HRT. The centralized algorithm from [13] determines the unique round $t(v)$ in which each node v transmits all messages from its subtree of T[4] to the parent of v without a collission. These collision-free transmissions are assured by the properties of *gathering trees* from [13] which are also satisfied by 2-HRT. The value of $t(v)$ depends on parameters D, *level*(v), $h_2(v) \in [0, \log n]$, Δ and on some auxiliary label $s(v) \in [0, \Delta - 1]$. Thus, while Δ and $s(v)$ can be encoded in the label of v using $O(\log \Delta)$ bits, we cannot store D, $h_2(v)$ and *level*(v) in the label of length $O(\log \Delta)$. With this aim we use a modified Size Learning Algorithm from [11] followed by an acknowledged broadcasting algorithm to share information about the value of D among all nodes and assure that nodes learn their levels during an execution of the broadcasting algorithm. Finally, each leaf is marked as such by an appropriate bit of its label. The fact that a node v is a leaf implies also that $h_2(v) = 0$. Other nodes learn their values of h_2 by modifying the maximal values of h_2 of their children.

Our solution of the gossiping problem works as follows. First, we gather all messages in an arbitrary node s, executing our gathering algorithm. Then, all messages collected at s are distributed using our broadcasting algorithm. In order to coordinate all nodes so that they know when the consecutive subroutines of the final algorithm start, we use an *acknowledged* broadcasting.

[4] The authors of [13] use the notion of gathering trees, but 2-HRT satisfy all properties of gathering trees.

References

1 Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM J. Discret. Math.*, 33(1):116–137, 2019. doi:10.1137/16M1105967.

2 Reuven Bar-Yehuda, Oded Goldreich, and Alon Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *J. Comput. Syst. Sci.*, 45(1):104–126, 1992. doi:10.1016/0022-0000(92)90042-H.

3 I. Chlamtac and S. Kutten. On broadcasting in radio networks–problem analysis and protocol design. *IEEE Trans. on Communic.*, 33(12):1240–1246, 1985. doi:10.1109/TCOM.1985.1096245.

4 Ferdinando Cicalese, Fredrik Manne, and Qin Xin. Faster deterministic communication in radio networks. *Algorithmica*, 54(2):226–242, 2009. doi:10.1007/S00453-007-9136-0.

5 Artur Czumaj and Peter Davies. Deterministic communication in radio networks. *SIAM J. Comput.*, 47(1):218–240, 2018. doi:10.1137/17M1111322.

6 Artur Czumaj and Wojciech Rytter. Broadcasting algorithms in radio networks with unknown topology. *J. Algorithms*, 60(2):115–143, 2006. doi:10.1016/J.JALGOR.2004.08.001.

7 Faith Ellen and Seth Gilbert. Constant-length labelling schemes for faster deterministic radio broadcast. In C. Scheideler and M. Spear, editors, *SPAA '20*, pages 213–222. ACM, 2020. doi:10.1145/3350755.3400238.

8 Faith Ellen, Barun Gorain, Avery Miller, and Andrzej Pelc. Constant-length labeling schemes for deterministic radio broadcast. In Christian Scheideler and Petra Berenbrink, editors, *SPAA 2019*, pages 171–178. ACM, 2019. doi:10.1145/3323165.3323194.

9 Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST computation with short advice. *Theory Comput. Syst.*, 47(4):920–933, 2010. doi:10.1007/s00224-010-9280-9.

10 Emanuele G. Fusco, Andrzej Pelc, and Rossella Petreschi. Topology recognition with advice. *Inf. Comput.*, 247:254–265, 2016. doi:10.1016/j.ic.2016.01.005.

11 A. Ganczorz, T. Jurdzinski, M. Lewko, and A. Pelc. Deterministic size discovery and topology recognition in radio networks with short labels. In S. Gilbert, editor, *DISC 2021*, volume 209 of *LIPIcs*, pages 22:1–22:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.DISC.2021.22.

12 Leszek Gasieniec and Andrzej Lingas. On adaptive deterministic gossiping in ad hoc radio networks. *Inf. Process. Lett.*, 83(2):89–93, 2002. doi:10.1016/S0020-0190(01)00312-X.

13 Leszek Gasieniec, David Peleg, and Qin Xin. Faster communication in known topology radio networks. *Distributed Comput.*, 19(4):289–300, 2007. doi:10.1007/s00446-006-0011-z.

14 Leszek Gasieniec, Tomasz Radzik, and Qin Xin. Faster deterministic gossiping in directed ad hoc radio networks. In Torben Hagerup and Jyrki Katajainen, editors, *SWAT 2004*, volume 3111 of *LNCS*, pages 397–407. Springer, 2004. doi:10.1007/978-3-540-27810-8_34.

15 Mohsen Ghaffari and Bernhard Haeupler. Fast structuring of radio networks large for multi-message communications. In Yehuda Afek, editor, *DISC 2013*, volume 8205 of *Lecture Notes in Computer Science*, pages 492–506. Springer, 2013. doi:10.1007/978-3-642-41527-2_34.

16 C. Glacet, A. Miller, and A. Pelc. Time vs. information tradeoffs for leader election in anonymous trees. *ACM Trans. Algorithms*, 13(3):31:1–31:41, 2017. doi:10.1145/3039870.

17 Barun Gorain and Andrzej Pelc. Deterministic graph exploration with advice. *ACM Trans. Algorithms*, 15(1):8:1–8:17, 2019. doi:10.1145/3280823.

18 Dariusz R. Kowalski and Andrzej Pelc. Broadcasting in undirected ad hoc radio networks. *Distributed Computing*, 18(1):43–57, 2005. doi:10.1007/s00446-005-0126-7.

19 Dariusz R. Kowalski and Andrzej Pelc. Optimal deterministic broadcasting in known topology radio networks. *Distributed Computing*, 19(3):185–195, 2007. doi:10.1007/s00446-006-0007-8.

20 Gianluca De Marco. Distributed broadcast in unknown radio networks. *SIAM J. Comput.*, 39(6):2162–2175, 2010. doi:10.1137/080733826.

21 Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theor. Comput. Sci.*, 608:190–198, 2015. doi:10.1016/J.TCS.2015.09.025.

22 Shailesh Vaya. Round complexity of leader election and gossiping in bidirectional radio networks. *Inf. Process. Lett.*, 113(9):307–312, 2013. doi:10.1016/J.IPL.2013.02.001.

Brief Announcement: Carry the Tail in Consensus Protocols

Suyash Gupta
University of Oregon, Eugene, OR, USA

Dakai Kang
University of California Davis, CA, USA

Dahlia Malkhi
University of California Santa Barbara, CA, USA

Mohammad Sadoghi
University of California Davis, CA, USA

Abstract

We present **Carry-the-Tail**, the first deterministic atomic broadcast protocol in partial synchrony that, after GST, simultaneously guarantees two desirable properties: (i) a constant fraction of commits are proposed by non-faulty leaders against tail-forking attacks, and (ii) optimal, worst-case quadratic communication under a cascade of faulty leaders. The solution also guarantees linear amortized communication, i.e., the steady-state is linear. Combining these two desirable properties was not simultaneously achieved previously: on one hand, prior atomic broadcast solutions achieve per-view linear word communication complexity. However, they face a significant degradation in throughput under tail-forking attack. On the other hand, existing solutions to tail-forking attacks require either quadratic communication steps or computationally-prohibitive SNARK generation.

The key technical contribution is **Carry**, a practical drop-in mechanism for streamlined protocols in the HotStuff family. Carry guarantees good performance against tail-forking and removes most leader-induced stalls, while retaining linear traffic and protocol simplicity. Carry-the-Tail implements the Carry mechanism on HotStuff-2.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Consensus, Blockchain, BFT

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.59

Related Version *Full Version*: `https://arxiv.org/abs/2508.12173` [6]

Funding *Mohammad Sadoghi*: This work is partially funded by NSF Award Number 2245373.

1 Introduction

In a "streamlined" Byzantine Fault Tolerant (BFT) consensus approach, which was pioneered by HotStuff [15], the consensus protocol has a simple and uniform structure: each *view* is a single quorum exchange between a leader and voters, and each such exchange carries a new leader proposal. Since a single quorum exchange does not suffice to achieve a consensus decision, leaders have to rely on the next leader to drive a second exchange that commits the previous proposal. HotStuff-like (HS-like) protocols [1, 2, 4, 9, 10, 13, 14, 15] are widely adopted in modern blockchain and decentralized systems for their conceptual simplicity, responsive liveness, linear communication, and censorship resistance.

BeeGees [5] exposed a vulnerability of the streamlined regime. In a *tail-forking* attack, a malicious or sluggish next leader can skip over the previous leader's proposal. Repeated attacks by bad leaders might significantly degrade throughput.


© Suyash Gupta, Dakai Kang, Dahlia Malkhi, and Mohammad Sadoghi;
licensed under Creative Commons License CC-BY 4.0



We present **Carry**, a lightweight, drop-in mechanism for HS-like protocols that defends against tail-forking. The mechanism also boosts performance under straggler leaders by ensuring safe progress with aggressive responsiveness without waiting for full quorums. Notably, Carry incurs a linear communication overhead per view.

Carry is a generic mechanism. We demonstrate its application to HotStuff-2 to create Carry-the-Tail, a full consensus solution that achieves the following two guarantees: ρ-tail-resilience (see Definition 1) and liveness under hostile or sluggish leadership with $O(n)$ communication per view.

Previously, no solution simultaneously achieved both properties. Existing tail-forking defenses either require expensive quadratic communication [5] or computationally heavy SNARKs to prove the absence of quorum certificates [7, 8].

▶ **Definition 1** (ρ-tail-resilience)**.** We say that a proposal T is *ρ-isolated* if the view of T is situated among a succession of ρ consecutive bad leaders. (After GST,) each proposal T by an honest leader, which is not ρ-isolated, is guaranteed to be included in the global sequence.

2 Overview of Carry-the-Tail

Carry is a generic liveness-boost for protocols in the HotStuff family. The setting for this work is described in the full paper [6]; briefly, the system consists of $N = 3F + 1$ replicas with up to F Byzantine faulty, or F_{actual} faulty in an actual execution, $F_{actual} \leq F$, and partially synchronous communication. A full-fledged protocol referred to as Carry-the-Tail is described here. It integrates the Carry approach into HotStuff-2.

Proposing and Voting

The Carry-the-Tail protocol operates in a view-by-view manner. Each view v has a known designated leader L_v performing a single quorum-exchange with replicas. The leader broadcasts a proposal B_v to the replicas; replicas respond (subject to safety rules) with a NEW-VIEW message carrying a signature-share on B_v. The signature-share is referred to as a *vote* on B_v.

A Quorum-Certificate (QC) consists of a threshold-signature by a quorum of $2F + 1$ replicas. We say that the proposal is *certified* when $\mathsf{QC}(B_v)$ is formed. To form QCs, leaders collect signature-shares for previous proposals and aggregate them. Broadcasting a QC incurs only linear word-communication complexity.

Commit Safety and Liveness

Each proposal B_v includes an opaque payload and meta-information. The meta-information references a history known to the leader. It includes $B_v.QC$ prior to view v, the highest known certified tail. Chaining proposals to one another using cryptographic certificates is utilized along with protocol voting and commit rules to ensure **safety**.

Figure 1 depicts a failure-free flow of the protocol with leader proposals chained to one another via QCs. Upon receiving a proposal B_v, a replica becomes *locked* on $B_v.QC$. Locking is key to ensuring safety: in the future, the replica pledges that it will only accept proposals that extend $B_v.QC$ or a QC from a higher view. The figure shows a failure-free execution snippet with four consecutive leaders chaining proposals one after another.

A *commit* necessitates two consecutive successful views. If a leader L_v forms $\mathsf{QC}(v-1)$ for the proposal B_{v-1} immediately preceding it, then the proposal becomes committed once there are $2F + 1$ signature-shares on B_v, forming $\mathsf{QC}(v)$. Anyone can learn this commit

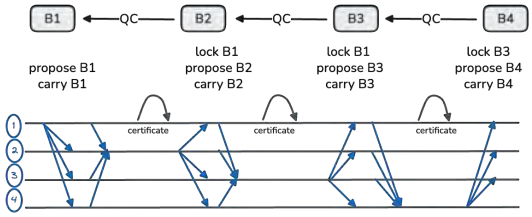


Figure 1 Basic flow in a failure-free scenario.

decision by obtaining these votes. In particular, anyone observing a chain $B_*.QC = \mathsf{QC}(v)$, $B_v.QC = \mathsf{QC}(v-1)$ learns that B_{v-1} has been committed. In Figure 1, B_1 and B_2 have been committed, and anyone observing this chain can learn these decisions.

To guarantee **liveness**, a leader proposes if it received a QC from the immediately-preceding view or a timeout. The timeout is implemented by a separate view-synchronization (Pacemaker) module [3, 11, 12]. Any (linear) Pacemaker can be plugged here. The Pacemaker guarantees that sufficient time has elapsed for all honest replicas to have entered view v and for their NEW-VIEW messages to have arrived at the leader.

The problem with Failed Views and Tail-Forking

We first explain the tail-forking attack which was exposed in BeeGees [5], and later describe how Carry prevents it.

Figure 2(a) depicts a scenario with failed view B_2, zooming in on the transition from B_2 to B_3. In view 2, no replica succeeds to vote for B_2. L_3, the leader of view 3, times out without having formed a QC for B_2 and must skip B_2.

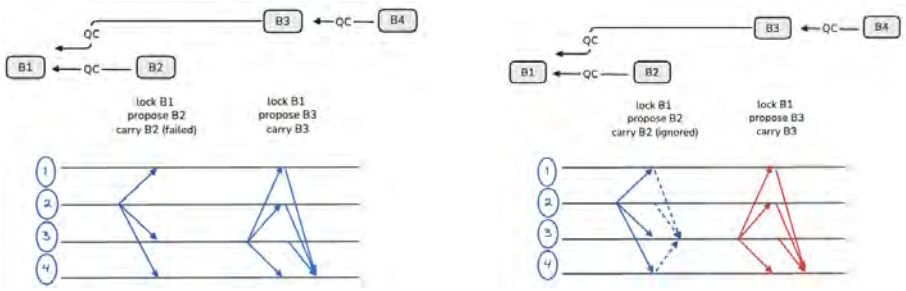
The problem is that a malicious L_3 could exploit this and omit votes for B_2 to wrongfully skip it. Figure 2(b) shows that B_3 could ignore the $2F+1$ honest votes (the dashed arrows) for B_2 and wrongfully skip it. More generally, a view v suffers a *tail-forking* attack if $2F+1$ signature shares are sent by honest replicas on B_v, but the next leader L_{v+1} intentionally ignores and skips it. The attack can be caused by either a malicious or a sluggish leader.

Regardless of the cause for tail-forking, it causes significant performance degradation. First, a proposal (e.g., B_2) is unnecessarily dropped. Second, the latency to a commit decision on pending earlier proposals increases. Moreover, tail-forking might occur frequently if there are many malicious/sluggish leaders. In the worst case, F bad leaders perfectly interspersed among honest views as depicted in Figure 4(perf1), where leaders $L_3, L_5, ...$, are bad. This might cause $O(N)$ throughput degradation and $O(N)$ latency increase.

As explained below, the Carry method prevents the tail-foring attacks from happening.

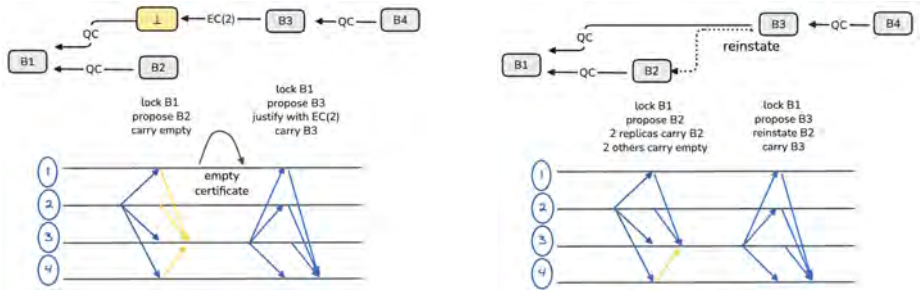
Protecting the Tail

Because replicas send NEW-VIEW messages to the next leader, in lieu of a vote, they can send a signature-share on an empty vote ("$\perp$"). In Carry, a *justified* skip over a tail B_v is allowed if it is accompanied by an *Empty Certificate (EC)* consisting of a threshold signature by $2F+1$ replicas on $\perp$. An *Empty Certificate* cannot possibly be formed if $F+1$ honest replicas voted for B_v; but if only F or fewer honest replicas voted for B_v, it could form, which affects neither safety nor liveness. Figure 3(c) depicts shows how *EC*s justifies a skipped tail, where L_3 collects $2F+1$ empty votes (the yellow arrows) for view 2 and forms an $\mathsf{EC}(2)$.



(a) (b)

Figure 2 Without tail protection: (a) B_2 failed and benignly skipped. (b) B_2 tail-forked.



(c) (d)

Figure 3 With Carry protection: (c) B_2 failed and EC-justified skipped. (d) B_2 reinstated.

Reinstating Uncertified Tails

If there is neither a QC nor an EC, previous solutions (e.g., [5]) required leaders to justify skipping the tail by using all the votes, resulting in cubic word-communication complexity in the worst case with F consecutive malicious leaders.

Instead, the main idea in Carry is to force the next leader to *reinstate* the last (possibly uncertified) voted block. To implement this, Carry introduces an optional field $B_v.reinstate$. This field references an uncertified earlier proposal B_x that becomes an integral part of B_v. As illustrated in Figure 3(d), L_3 receives $F+1 = 2$ votes for B_2 and two empty votes, preventing the formation of either $\mathsf{QC}(2)$ or $\mathsf{EC}(2)$. Consequently, B_2 is *reinstated* in B_3.

With this new mechanism, as shown in Figure 4(perf2), a leader must justify skipping a tail via an EC or else reinstate the tail. Notably, reinstating also helps if insufficient votes arrive for B_v due to mere sluggishness.

ρ-Tail-Resilience

Reinstating protects the tail in case an EC cannot be formed on an immediately preceding view. Unfortunately, tail-forking of B_v remains possible by having two consecutive malicious leaders following B_v. Attacks by two consecutive malicious leaders, L_{v+1} and L_{v+2}, are depicted in Figure 4(perf3), e.g., L_3 and L_4. First, L_{v+1} "fails" in view $v+1$, not forming $\mathsf{QC}(v)$. Next, B_{v+2} uses $\mathsf{EC}(v+1)$ to justify skipping B_{v+1}. Furthermore, B_{v+2} skips B_v because no justification is required to skip B_v and no replica is locked on it.

To protect against tail-forking by two or more consecutive bad leaders, we apply the Carry method to the last ρ views. The NEW-VIEW messages from replicas should carry their votes, possibly empty, for the *last* ρ *views*. This protects a tail unless it is followed by ρ consecutive

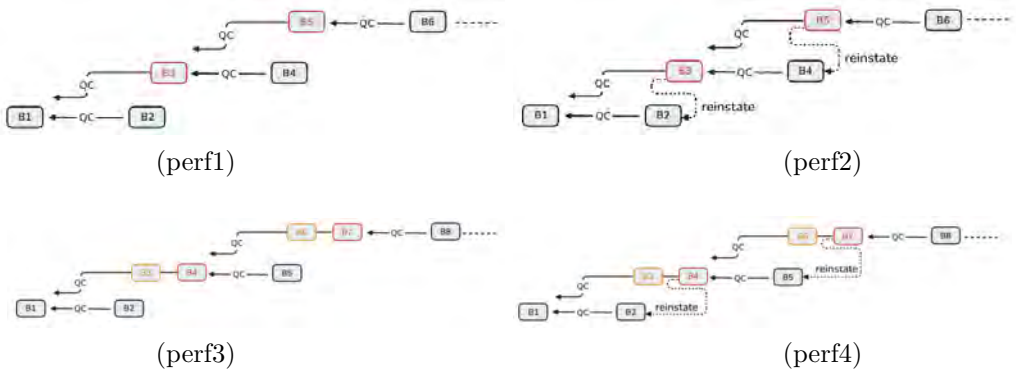


Figure 4 (perf1) Performance Degradation due to Tail-Forking. (perf2) Improvement with Reinstating. (perf3) Degradation under Two-Leader Attacks. (perf4) Improvement with ρ-Tail-Resilience.

bad views. Hence, in each view rotation, all but (F_{actual}/ρ) proposals are protected from tail-forking. This property is referred to as $\rho{-}Tail{-}Resilience$. $2{-}Tail{-}Resilience$ is depicted in Figure 4(perf4). Importantly, for modest values of ρ, the chance of incurring ρ consecutive bad views in practice is very small, especially if leader rotation is randomized.

The overhead incurred is $O(\rho N)$ word communication.

Comparison with AHL

BeeGees formulated a property called *"Any-Honest-Leader commit"* (AHL): after GST, once an honest leader proposes in a view, that block will be committed after at most k subsequent honest-leader views[1]. BeeGees employs a complex leader handover in order to satisfy AHL (in one variant, it incurs quadratic word-communication per view, and in a second variant, it relies on computationally-prohibitive SNARK proofs for compressing communication).

The property *ρ-tail-resilience* guarantees, under a reasonably small choice of ρ (e.g. $\rho = 6$), that a large constant fraction of honest leader proposals become committed even against the worst-case tail-forking attacks. It is worth noting that by setting $\rho = f$, we get the same AHL property in Carry as in BeeGees, while incurring the same communication burden. However, arguably Carry has simpler logic and also allows ρ to be a tunable parameter in production.

3 Carry

In this section, we show how Carry is implemented, including the *Empty Certificate Justification* and *Reinstate* mechanisms. Replicas carry their votes (which are possibly empty) for the last ρ views in their NEW-VIEW messages, and the new leader must reinstate the last uncertified proposal.

[1] k is a protocol parameter indicating the number of phases to reach a commit; typically, $k = 2$ or $k = 3$.

Carry Implementation

Carry Rule. A valid proposal B_v from the leader L_v of view v has the following format:

Highest QC: the highest quorum certificate $\mathsf{QC}(x)$ known to L_v is attached as $B_v.qc = \mathsf{QC}(x)$;

Reinstate: if L_v received fewer than $2F+1$ votes for the highest tail T extending B_x, and $v - x \leq \rho$, then T is reinstated (in full) as $B_v.reinstated = T$;

Justification: If $v - x \leq \rho$, then empty-certificates are attached to B_v for each view between x and v, which B_v does **not** extend. It is worth noting that if $B_v.reinstated$ exists, empty-certificates for views preceding the reinstated block are recursively attached within the chain of reinstated blocks.

Voting Rule. A replica accepts and votes for the proposal B_v from leader L_v of view v if:

1. $B_v.qc$ has a higher or equal view than the replica's *lock*,
2. B_v adheres to the Carry Rule above.

Carry retains from HotStuff-2 the Commit Rule and Leader Proposal Rule.

3.1 The Carry-the-Tail Protocol

Akin to HotStuff-2, the Carry-the-Tail protocol flows view-by-view. At the end of each view, the replicas and the incoming leader perform a handover protocol as follows [2]:

Replica ⟶ incoming leader. Each replica sends an incoming leader a `NEW-VIEW` message that carries

1. the next view number
2. the replica's highest QC (its *lock*)
3. its votes (possibly empty) in the past ρ views.[3]

Leader ⟶ replicas. On satisfying the Leader Proposal Rule, the leader of view v proposes a block B_v that

1. extends the highest QC it has collected, potentially freshly aggregated from `NEW-VIEW` messages, as $B_v.qc$,
2. reinstates T in full as $B_v.reinstated$, provided that the highest QC is from the past ρ views and the highest voted block T extends the highest QC,
3. attaches empty-certificates for each view between view(T) and v.

References

1 Jeb Bearer, Benedikt Bünz, Philippe Camacho, Binyi Chen, Ellie Davidson, Ben Fisch, Brendon Fish, Gus Gutoski, Fernando Krell, Chengyu Lin, et al. The espresso sequencing network: Hotshot consensus, tiramisu data-availability, and builder-exchange. *Cryptology ePrint Archive*, 2024.

2 Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. *IACR Cryptol. ePrint Arch.*, 2023:463, 2023. URL: `https://api.semanticscholar.org/CorpusID:259092405`.

[2] See full version [6] for detailed design, pseudocode and correctness proofs.

[3] It is possible to send less information if the lock precedes the current view by less than ρ views; we omit this optimization for simplicity.

3 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Byzantine consensus is $\theta(n^2)$: The dolev-reischuk bound is tight even in partial synchrony! In *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:21. Schloss Dagstuhl, 2022. doi:10.4230/LIPIcs.DISC.2022.14.

4 Diem. DiemBFT consensus protocol, 2020. URL: https://github.com/diem/diem/tree/latest/consensus.

5 Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. BeeGees: Stayin' alive in chained BFT. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, PODC '23, pages 233–243, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3583668.3594572.

6 Suyash Gupta, Dakai Kang, Dahlia Malkhi, and Mohammad Sadoghi. Carry the tail in consensus protocols, August 2025. doi:10.48550/arXiv.2508.12173.

7 Mohammad Mussadiq Jalalzai and Kushal Babel. Monadbft: Fast, responsive, fork-resistant streamlined consensus. *ArXiv*, abs/2502.20692, 2025. doi:10.48550/arXiv.2502.20692.

8 Mohammad Mussadiq Jalalzai, Chen Feng, and Victoria Lemieux. Vbft: Veloce byzantine fault tolerant consensus for blockchains. *ArXiv*, abs/2310.09663, 2023. doi:10.48550/arXiv.2310.09663.

9 Dakai Kang, Suyash Gupta, Dahlia Malkhi, and Mohammad Sadoghi. Hotstuff-1: Linear consensus with one-phase speculation. *Proceedings of the ACM on Management of Data*, 3(3):1–29, 2025. doi:10.1145/3725308.

10 Dakai Kang, Sajjad Rahnama, Jelle Hellings, and Mohammad Sadoghi. SpotLess: Concurrent rotational consensus made practical through rapid view synchronization. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, Netherlands, May 13-17, 2024*. IEEE, 2024. doi:10.1109/ICDE60146.2024.00157.

11 Andrew Lewis-Pye. Quadratic worst-case message complexity for state machine replication in the partial synchrony model, 2022. arXiv:2201.01107.

12 Andrew Lewis-Pye, Dahlia Malkhi, Oded Naor, and Kartik Nayak. Lumiere: Making optimal BFT for partial synchrony practical. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 135–144, 2024. doi:10.1145/3662158.3662787.

13 Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft, 2023. URL: https://api.semanticscholar.org/CorpusID:259144145.

14 Jianyu Niu, Fangyu Gai, Mohammad M. Jalalzai, and Chen Feng. On the performance of pipelined hotstuff. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021. doi:10.1109/INFOCOM42981.2021.9488706.

15 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019. URL: https://api.semanticscholar.org/CorpusID:197644531.

Brief Announcement: Distributed Sparsest Cut via Eigenvalue Estimation

Yannic Maus[1]
TU Graz, Austria

Tijn de Vos
TU Graz, Austria

Abstract

We give new, improved bounds for approximating the sparsest cut value or in other words the conductance ϕ of a graph in the CONGEST model. As our main result, we present an algorithm running in $O(\log^2 n/\phi)$ rounds in which every vertex outputs a value $\tilde{\phi}$ satisfying $\phi \leq \tilde{\phi} \leq \sqrt{2.01\phi}$. In most regimes, our algorithm improves significantly over the previously fastest algorithm for the problem [Chen, Meierhans, Probst Gutenberg, Saranurak; SODA 25]. Additionally, our result generalizes to k-way conductance.

We obtain these results, by approximating the eigenvalues of the normalized Laplacian matrix $L := I - \mathrm{Deg}^{-1/2} A \mathrm{Deg}^{-1/2}$, where, A is the adjacency matrix and Deg is the diagonal matrix with the weighted degrees on the diagonal. We show our algorithms are near-optimal by proving a lower bound for computing the smallest non-trivial eigenvalue of L, even in the stronger LOCAL model

The previous state of the art sparsest cut algorithm is in the technical realm of expander decompositions. Our algorithms, on the other hand, are relatively simple and easy to implement. At the core, they rely on the well-known *power method*, which comes down to repeatedly multiplying the Laplacian with a vector. This operation can be performed in a single round in the CONGEST model.

All our algorithms apply to weighted, undirected graphs. Our lower bounds apply even in unweighted graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Distributed algorithms

Keywords and phrases CONGEST, Sparsest Cut, Laplacian, Eigenvalues, Spectral Graph Theory

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.60

Related Version *Full Version*: `http://arxiv.org/abs/2508.19898`

Funding This research was funded in whole or in part by the Austrian Science Fund (FWF) `https://doi.org/10.55776/P36280` and `https://doi.org/10.55776/I6915`. For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

Acknowledgements We would like to thank Leo Wennmann, for the many hours of discussion on this topic. We would like Joachim Orthaber and Malte Baumecker, for thinking along. We would like to thank Faith Ellen, Sebastian Brandt, Alexandre Nolin, and Eva Rotenberg, for the initial discussions that were the inspiration for this research. We would like to thank Schloss Dagstuhl, since the research was initiated during the Dagstuhl seminar *Graph Algorithms: Distributed Meets Dynamic*.

[1] The author ordering was randomized using `https://www.aeaweb.org/journals/policies/random-author-order/` generator. It is requested that citations of this work list the authors separated by ⓡ instead of commas.



© Yannic Maus and Tijn de Vos;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

A graph cut is a partition of the vertices into two disjoint subsets, where the size of the cut is the total number of edges crossing between them, and the sparsity of the cut measures how small this cut is relative to total edge weight of the smaller side, i.e., in comparison to the sum of its vertex degrees. Graph cuts are a central object in theoretical computer science and algorithm design. Cuts are especially important in communication networks because they represent the capacity for information to flow between different parts of the system. A sparse cut – where only a few edges connect two large subsets – can become a bottleneck, severely limiting bandwidth or throughput. A central question for guaranteeing the performance of network algorithms is to determine the minimum sparsity of cuts in a given communication network. For example the runtime of the famous PUSH-PULL protocol depends on the sparsest cut value [11]. More generally, graphs without sparse cuts support fast mixing of random walks [20], and enable efficient routing due to their strong connectivity [9, 10]. This motivates the use of expander decompositions, which partition a graph into *high-conductance* clusters (i.e, no sparse internal cuts), and sparse inter-cluster connections. Such decompositions are powerful algorithmic tools: thanks to efficient routing, many algorithms run significantly faster on high-conductance subgraphs, allowing problems to be solved locally within clusters before handling the sparse inter-cluster parts.

In this work, we present simple, time and bandwidth efficient distributed algorithms to approximate the sparsest cut value. Our randomized algorithms run in the classic CONGEST model of distributed computing where a communication network is abstracted as an n-vertex graph with vertices being computing entities and edges serving as bandwidth-limited communication links. Communication happens in synchronous rounds, in each of which each vertex can send a $O(\log n)$-bit message to each of its neighbors and the complexity measure is the number of communication rounds until the vertices have computed their outputs, e.g., until they have output an approximate value of the sparsest cut.

Formally, the *sparsity* of a set (or cut) $S \subseteq V$ in a graph $G = (V, E)$ is defined as as

$$\phi(S) := \frac{|E(S, V \setminus S)|}{\min\{\mathrm{Vol}(S), \mathrm{Vol}(V \setminus S)\}} ,$$

where the volume $\mathrm{Vol}(S)$ of a set of vertices S is the sum of of the degrees of the vertices in the set $\sum_{v \in S} \deg(v)$. The value $\phi(S)$ is also known as the *conductance* of the set S. The *sparsest cut* $\phi(G) \in [0, 1]$ of a graph G is the minimum sparsity of a subset $S \subseteq V$. This is also known as the *conductance* of the graph.

The relative notion $\phi(G)$ can be viewed as a normalized quantification of cut size. In contrast to computing minimum cuts, which quantify the absolute size of a cut, determining the sparsest cut or the conductance is not only NP-complete [17] but also APX-hard under the unique games conjecture [6]. Consequently, although no polynomial-time algorithm can approximate the optimum arbitrarily well (unless $P = NP$), a variety of approximation algorithms with provable – albeit weaker – guarantees have been developed across different computational models. Next, we present our results and compare them with the state of the art in the field. As the sparsest cut and the conductance have been extensively studied we provide further background and related work in the full version.

2 Our Contribution on Sparsest Cut Approximation

Our first result is an efficient CONGEST algorithm to approximate the sparsest cut.

▶ **Theorem 1.** *There is a randomized* CONGEST *algorithm that, given an undirected weighted n-node graph with sparsest cut ϕ, w.h.p. gives an approximation of the sparsest cut in $O(\log^2 n/\phi)$ rounds. In particular, every vertex outputs $\tilde{\phi}$ such that $\phi \leq \tilde{\phi} \leq \sqrt{2.01\phi}$.*

In general, there are two approaches for developing approximation algorithms for the sparsest cut, the *flow approach*, that we detail in the full version, and the *spectral approach*, which we use to prove Theorem 1. On a high-level, one computes the spectrum (or parts of it) of the graph, i.e., its eigenvalues, and then uses Cheeger's Inequality[2] to derive an approximation of $\phi(G)$ from the eigenvalues. Due to the limitations imposed by Cheeger's Inequality, the best possible approximation achievable through spectral methods is $\sqrt{2\phi}$. Our technical result is slightly stronger than Theorem 1. In fact, we can attain this theoretical bound – up to an arbitrarily small additive error – in $O(\log^2 n/\varepsilon^2 + \log n/\varepsilon^4)$ rounds. See the proof of Theorem 1 for the details.

For unweighted graphs, we compare our result against the previously best algorithm by Chen, Meierhans, Probst Gutenberg, and Saranurak that provides a $O(\phi \log^2 n)$-approximation of ϕ [7][3]. Our approximation is more precise than [7] when $\phi = \Omega(1/\log^4 n)$, and is faster in all cases: [7] takes $O(\operatorname{poly}\log n/\phi^4)$ rounds. Upon inspection of their $\operatorname{poly}\log n$, they have a total round complexity of at least $O(\log^{20} n)$ in the regime where their approximation is better.

Prior to [7], Das Sarma, Molla, and Pandurangan [19] gave a different algorithm to approximate the sparsest cut, which also works for weighted graphs. They computed an $O(\sqrt{\phi}\operatorname{poly}\log n)$-approximation in $O(\frac{1}{b}(n+\frac{1}{\phi})\log^2 n)$ rounds if the graph has a sparse cut of *balance* b, i.e., both sides S of the cut satisfy $\operatorname{Vol}(S) \geq 2b|E|$. Building on the methods of [19], Kuhn and Molla improved the approximation ratio to $O(\sqrt{\phi \log n})$ and the runtime to $O(D + \frac{\log^2 n}{b\phi})$ [14]. For balanced cuts $b = \Omega(1)$, we improve their approximation ratio by a factor $\sqrt{\log n}$, while matching their running time. For unbalanced cuts, also their algorithm can take as much as $\Omega(n^2)$ time, so we do not only improve the approximation ratio, but also the running time exponentially.

k-Way Sparsest Cut. There exists a natural generalization of the sparsest cut to *multi-cuts* [15]: the *k-way sparsest cut* or *k-way conductance* ϕ_k is defined follows:

$$\phi_k := \min_{V_1,V_2,\dots,V_k \subset V} \max_{i=1,\dots,k} \frac{|E(V_i, V \setminus V_i|}{\operatorname{Vol}(V_i)},$$

where the minimum is taken over disjoint, non-empty V_i. Note that $\phi_2 = \phi$. The k-way conductance captures how well-connected a graph is across multiple parts; it can describe multi-partition bottlenecks, which standard conductance may miss. This is in particular relevant for real-world networks that usually consists of multiple clusters.

We prove the following theorem for approximating the k-way sparsest cut.

[2] Cheeger's Inequality [2, 1, 21] relates the second-smallest eigenvalue λ_2 of the normalized Laplacian to the graph's conductance ϕ, showing that $\frac{1}{2}\lambda_2 \leq \phi \leq \sqrt{2\lambda_2}$.

[3] They do not state this result explicitly, but they implement the cut-matching game from [18] (see Appendix A) in the distributed setting. As stated in [18], this cut-matching game gives such an approximation.

▶ **Theorem 2.** *There is a randomized* CONGEST *algorithm that, given a constant $k \geq 2$ and undirected weighted graph with k-way sparsest cut ϕ_k, w.h.p. gives an approximation of the k-way sparsest cut in $O(\log^2 n \operatorname{poly}(\phi_k^{-1}))$ rounds. In particular, every vertex outputs $\tilde{\phi}_k$ such that $\phi_k \leq \tilde{\phi}_k \leq O(\sqrt{\phi_k})$.*

To the best of our knowledge, we are the first to study the k-way conductance in the distributed setting.

Lower bounds on approximating sparse cuts. Generally, it is known that approximating the sparsest cut takes $\Omega(D)$ rounds. In particular, Das Sarma, Molla, and Pandurangan [19] show a $\tilde{\Omega}(D + \sqrt{n})$ CONGEST lower bound for any multiplicative approximation in weighted graphs. They reduce the spanning connected subgraph problem to a sparsest cut instance with edge weights that are either 0 or 1. This means that the lower bound shows that it is hard to decide whether $\phi = 0$ holds or not. For ϕ bounded away from zero, there is no such lower bound. Our upper bound of $O(\log^2 n/\phi)$ rounds, Theorem 1, indeed takes D rounds, since $D = O(\log n/\phi)$. However, for $\phi = \omega(1/\sqrt{n})$, the upper bound does not have a $\sqrt{n}$-term.

We prove a $\Omega(D)$ lower bound that holds even for unweighted graphs and in the LOCAL model. LOCAL is identical to CONGEST, except that there is no bound on the message size. Even for a class of fixed diameter, any algorithm needs $\Omega(D)$ time for a $O(n)$-approximation.

▶ **Theorem 3.** *Let $D \leq n/2$ be a parameter and consider the class of graphs with diameter D. An algorithm that with probability $> \frac{1}{2}$ satisfies that every vertex v of a graph G outputs a value $\tilde{\phi}_v \geq \phi(G)$, for sparsest cut $\phi(G)$, and at least one vertex has $\phi_v \leq \Theta(n \cdot \phi)$ needs $\Omega(D)$ rounds in* LOCAL.

Note that the $\Omega(D)$ lower bound in Theorem 3 holds even in the case where only a single vertex needs to witness the cut.

The above proof is about discerning graphs with sparsity $1/n$ and $1/n^2$. If *every vertex* needs to output a correct estimation, a diameter lower bound exists for larger values of ϕ.

▶ **Theorem 4.** *Let $0 < \varepsilon < 1$ be a constant and let $\phi > 1/n^{1-\varepsilon}$. An algorithm that with probability $> \frac{1}{2}$ satisfies that every vertex v outputs a value $\tilde{\phi}_v$ such that $\phi \leq \tilde{\phi} \leq O(n^{1+\varepsilon} \cdot \phi)$, where ϕ is the sparsest cut value of the graph needs at least $\Omega(D)$ rounds in the* LOCAL *model.*

Our CONGEST upper bound for approximating the sparsest cut w.h.p. is $O(\log^2 n/\phi)$ – only a factor $\log n$ away from optimal, since the graphs from the above observation have sparsest cut ϕ and diameter $\Theta(\log n/\phi)$ [8]. We note that with constant probability, we match the lower bound, see the full version for details.

3 Our Contributions on Spectrum Approximation

Our core technical contributions are efficient CONGEST algorithms to compute the approximate (partial) spectrum of a graph. In essence, Theorems 1 and 2 then follow as direct consequences via Cheeger's Inequality. Nevertheless, we chose to present these theorems first, as they are more accessible to a broader audience and can be compared with prior work.

Eigenvalues of the Normalized Laplacian. The normalized Laplacian of a graph is the matrix defined as $L := I - \operatorname{Deg}^{-1/2} A \operatorname{Deg}^{-1/2}$, where A is the adjacency matrix and Deg is the (weighted) degree matrix. Its spectrum captures many central properties of the graph,

e.g., various types of its connectedness like k-way conductance [15] and small set expansion [3], or how close it is to being bipartite [4, 16]. We provide the following theorem to compute an approximation of that spectrum.

▶ **Theorem 5** (Eigenvalue estimation). *Let $\varepsilon > 0$ be a parameter and $k \in \mathbb{N}$ be a constant. There are randomized* CONGEST *algorithms that, given an undirected, weighted graph $G = (V, E)$, with high probability approximate the eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ of the graph's normalized Laplacian as follows:*

1. *$\lambda_1, \ldots, \lambda_k$: an additive ε-approximations in $O(\log^2 n \cdot \text{poly}(\varepsilon^{-1}))$ rounds,*
2. *λ_2: an additive ε-approximation in $O\left(\frac{\log^2 n}{\varepsilon} + \frac{\log n}{\varepsilon} \cdot \lceil \frac{\lambda_2}{\varepsilon} \rceil\right) = O\left(\frac{\log^2 n}{\varepsilon} + \frac{\log n}{\varepsilon^2}\right)$ rounds,*
3. *λ_n: a multiplicative $(1 \pm \varepsilon)$-approximation in $O\left(D + \frac{\log^2 n}{\varepsilon}\right)$ rounds.*

Eigenvalue λ_2 is the eigenvalue approximating the sparsest cut via Cheeger's Inequality. It could either be approximated via the second algorithm, or slightly slower via the first.

For the smaller eigenvalues, Theorem 5 states additive approximations avoiding the diameter dependence, e.g., when ε is an arbitrarily small constant Theorem 5 approximates λ_2 in $O(\log^2 n)$ rounds even though the diameter may be up to linear in n.

As ε can be arbitrary in Theorem 5, we can obtain multiplicative approximations by setting $\varepsilon = \varepsilon' \lambda_2$ for any $\varepsilon' > 0$. However, this does not circumvent the diameter lower bound of Theorem 6 presented below. $D = O(\log n / \lambda_2)$ holds for any graph, so the diameter term is then always asymptotically smaller than the overall runtime of the algorithm.

To the best of our knowledge, the only prior distributed eigenvalue computation is via computing a spectral sparsifier, i.e., a sparse subgraph H with $\tilde{O}(n/\varepsilon^2)$ edges that approximately preserve the graph's spectrum. Next, the sparsifier H can be gathered at one vertex of the graph that can locally compute the eigenvalues of H and hence obtain approximations of all eigenvalues of G. Computing H is actually relatively fast and only requires $\text{poly}(\log n, \varepsilon^{-1})$ rounds [13]. Collecting H at a single vertex is the expensive part, requiring $\tilde{O}(n/\varepsilon^2)$ rounds. This is exponentially slower than Theorem 5, though this more general method immediately computes the full spectrum while we focus on single eigenvalues.

We note that all our upper bounds even hold in the more restricted Broadcast CONGEST model in which a vertex needs to send the same message to all of its neighbors in each round. All our algorithms become a $\log n$-factor faster if we only ask for constant probability as typical in property testing, see the discussion in the full version.

Lower Bound. All of our multiplicative upper bounds take diameter time. This means that at the end of the algorithm, every vertex can easily output the same eigenvalue or sparsest cut estimate. One may ask whether one can obtain faster algorithms if the output is more local. Perhaps every vertex outputs the sparsest cut it sees in some neighborhood, and there is a guarantee that at least one vertex will see an approximately sparsest cut. In Theorem 3, we showed that this is not true for sparsest cut approximation. We also show that approximating λ_2 at even one node requires $\Omega(D)$ rounds.

▶ **Theorem 6.** *Any algorithm that with probability $> \frac{1}{2}$ ensures that all vertices output a value $\tilde{\lambda}_2 \geq \lambda_2$, and at least one vertex additionally satisfies $\tilde{\lambda}_2 \leq 2\lambda_2$ requires $\Omega(D)$ rounds.*

An $\Omega(D)$ lower bound could also follow from our sparsest cut lower bounds via Cheeger's Inequality, but such an approach incurs an inherent quadratic loss due to the approximation gap in Cheeger's.

4 Techniques: The Power Method

As mentioned, Theorems 1 and 2 follow as direct consequences from our eigenvalue estimation results via Cheeger's Inequality. Here, we focus on our core technical contribution: efficient CONGEST algorithms to compute the approximate (partial) spectrum of a graph. The main ingredient of our algorithms is the well-known *power method* for approximating the largest eigenvalue of a matrix. Let M be any matrix. The power method, also known as *power iteration* or *Von Mises iteration*, starts with a random vector and converges to the eigenvector corresponding to the largest eigenvalue of M. It is a standard technique in numerical linear algebra, see, e.g., [12]. The procedure is as follows:

Power Method

1: **Input:** Matrix $M \in \mathbb{R}^{n \times n}$, number of iterations k.
2: **Initialize:** A random vector $x_0 \in \mathbb{R}^n$.
3: **for** $i = 1$ to k **do**
4: $\quad x_i \leftarrow M x_{i-1}$.
5: **Output:**
- Approximate dominant eigenvalue: $\lambda \approx \frac{x_k^T M x_k}{x_k^T x_k}$;
- Approximate dominant eigenvector: x_k.

With constant probability over the randomness of the start vector x_0, the power method outputs a vector x_k whose *Rayleigh coefficient* $\frac{x_k^T M x_k}{x_k^T x_k}$ is close to the *largest* eigenvalue of the matrix M. In the full version, we provide both intuition and a formal proof.

In community detection, [5] used an algorithm that is in the same spirit as the power method but not entirely equal. In case of a sufficiently small balanced cut, and some additional technical assumptions, they can provide an approximate sparsest cut. Since the approximation factor depends on the technical assumptions, we will not state their exact guarantees here.

In the full version, we describe the power method in detail, provide a proof, and discuss the challenges of implementing it in a distributed setting.

References

1 Noga Alon. Eigenvalues and expanders. *Comb.*, 6(2):83–96, 1986. doi:10.1007/BF02579166.

2 Noga Alon and V. D. Milman. lambda$_1$, isoperimetric inequalities for graphs, and superconcentrators. *J. Comb. Theory B*, 38(1):73–88, 1985. doi:10.1016/0095-8956(85)90092-9.

3 Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *J. ACM*, 62(5):42:1–42:25, 2015. Announced at FOCS '10. doi:10.1145/2775105.

4 Frank Bauer and Jürger Jost. Bipartite and neighborhood graphs and the spectrum of the normalized graph laplacian. *Communications in Analysis and Geometry*, 21, October 2009. doi:10.4310/CAG.2013.v21.n4.a2.

5 Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Find your place: Simple distributed algorithms for community detection. *SIAM J. Comput.*, 49(4):821–864, 2020. Announced at SODA'17. doi:10.1137/19M1243026.

6 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006. Announced at CCC'05. doi:10.1007/S00037-006-0210-9.

7 Daoyuan Chen, Simon Meierhans, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Parallel and distributed expander decomposition: Simple, fast, and near-optimal. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 1705–1719. SIAM, 2025. doi:10.1137/1.9781611978322.53.

8 Flavio Chierichetti, George Giakkoupis, Silvio Lattanzi, and Alessandro Panconesi. Rumor spreading and conductance. *J. ACM*, 65(4):17:1–17:21, 2018. doi:10.1145/3173043.

9 Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 131–140. ACM, 2017. doi:10.1145/3087801.3087827.

10 Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.DISC.2018.31.

11 George Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, volume 9 of *LIPIcs*, pages 57–68. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICS.STACS.2011.57.

12 Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

13 Ioannis Koutis and Shen Chen Xu. Simple parallel and distributed algorithms for spectral graph sparsification. *ACM Trans. Parallel Comput.*, 3(2):14:1–14:14, 2016. doi:10.1145/2948062.

14 Fabian Kuhn and Anisur Rahaman Molla. Distributed sparse cut approximation. In Emmanuelle Anceaume, Christian Cachin, and Maria Gradinariu Potop-Butucaru, editors, *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, volume 46 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICS.OPODIS.2015.10.

15 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multi-way spectral partitioning and higher-order cheeger inequalities. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1117–1130. ACM, 2012. doi:10.1145/2213977.2214078.

16 Shiping Liu. Multi-way dual cheeger constants and spectral bounds of graphs. *Advances in Mathematics*, 268:306–338, 2015. arXiv:1401.3147.

17 David W. Matula and Farhad Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discret. Appl. Math.*, 27(1-2):113–123, 1990. doi:10.1016/0166-218X(90)90133-W.

18 Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635. SIAM, 2019. doi:10.1137/1.9781611975482.162.

19 Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Distributed computation of sparse cuts via random walks. In Sajal K. Das, Dilip Krishnaswamy, Santonu Karkar, Amos Korman, Mohan J. Kumar, Marius Portmann, and Srikanth Sastry, editors, *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 6:1–6:10. ACM, 2015. doi:10.1145/2684464.2684474.

20 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2:1–2:31, 2013. doi:10.1145/2432622.2432624.

21 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989. doi:10.1016/0890-5401(89)90067-9.

Brief Announcement: Asynchronous Approximate Agreement with Quadratic Communication

Mose Mizrahi Erbes
ETH Zürich, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

We consider an asynchronous network of n message-sending parties, up to t of which are byzantine. We study approximate agreement, where the parties obtain approximately equal outputs in the convex hull of their inputs. In their seminal work, Abraham, Amit and Dolev [OPODIS '04] solve this problem in $\mathbb{R}$ with the optimal resilience $t < \frac{n}{3}$ with a protocol where each party reliably broadcasts a value in every iteration. This takes $\Theta(n^2)$ messages per reliable broadcast, or $\Theta(n^3)$ messages per iteration.

In this work, we forgo reliable broadcast to achieve asynchronous approximate agreement against $t < \frac{n}{3}$ faults with quadratic communication. In a tree with the maximum degree Δ and the centroid decomposition height h, we achieve edge agreement in at most $6h + 1$ rounds with $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log \Delta + \log h)$ per round. We do this by designing a 6-round multivalued 2-graded consensus protocol and using it to recursively reduce the task to edge agreement in a subtree with a smaller centroid decomposition height. Then, we achieve edge agreement in the infinite path $\mathbb{Z}$, again with the help of 2-graded consensus. Finally, we show that our edge agreement protocol enables ε-agreement in $\mathbb{R}$ in $6 \log_2 \frac{M}{\varepsilon} + \mathcal{O}(\log \log \frac{M}{\varepsilon})$ rounds with $\mathcal{O}(n^2 \log \frac{M}{\varepsilon})$ messages and $\mathcal{O}(n^2 \log \frac{M}{\varepsilon} \log \log \frac{M}{\varepsilon})$ bits of communication, where M is the maximum non-byzantine input magnitude.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Approximate agreement, byzantine fault tolerance, communication complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.61

Related Version *Full Version*: `https://arxiv.org/abs/2408.05495` [10]

1 Introduction

We consider a fully connected asynchronous network of n message-passing parties $P_1, \ldots, P_n$. Up to t of these parties are corrupted in a byzantine manner, while the rest are honest.

In an approximate (convex) agreement problem, the parties output approximately equal values in the convex hull of their inputs. The most classical example is approximate agreement in $\mathbb{R}$, where the inputs/outputs are in $\mathbb{R}$, and for some parameter $\varepsilon > 0$ the following hold:

- ***validity:*** Each honest party output is between the minimum and maximum honest inputs.
- ***ε-agreement:*** If any honest parties P_i and P_j output y_i and y_j, then $|y_i - y_j| \leq \varepsilon$.

Approximate agreement in $\mathbb{R}$ was introduced in 1985 by Dolev, Lynch, Pinter, Stark and Weihl [8]. Like byzantine agreement, in synchronous networks it is possible against $t < \frac{n}{2}$ faults with setup [12], but only possible when $t < \frac{n}{3}$ if the network is asynchronous [1] or if perfect (signature-free) security is desired [8]. What sets approximate agreement apart is that it is determinism-friendly. While deterministic byzantine agreement takes $t+1$ rounds in synchrony [7] and is impossible against just one crash in asynchrony [11], approximate agreement does not share these limitations. Thus, approximate agreement protocols are customarily deterministic.

In [8], Dolev et al. achieve ε-agreement in $\mathbb{R}$ with a perfectly secure synchronous protocol secure against $t < \frac{n}{3}$ corruptions. Simplifying things slightly, in their protocol the parties estimate the spread S of their inputs (the maximum difference between any two inputs), and


© Mose Mizrahi Erbes and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0



run for $\lceil \log_2 \frac{S}{\varepsilon} \rceil$ rounds. In each round each party sends its value to every other party, and with this the parties halve the diameter of their values. After $\lceil \log_2 \frac{S}{\varepsilon} \rceil$ rounds, the spread is at most $2^{-\lceil \log_2(S/\varepsilon) \rceil} \leq \frac{\varepsilon}{S}$ of what it initially was, and thus ε-agreement is achieved. They present an asynchronous version of this protocol as well, but only with the resilience $t < \frac{n}{5}$ as the parties can no longer wait to receive the value of each honest party in every iteration.

Asynchronous approximate agreement in $\mathbb{R}$ with the optimal resilience $t < \frac{n}{3}$ was first achieved in 2004 by Abraham, Amit and Dolev [1], with a protocol that consists of $\mathcal{O}(\log \frac{S}{\varepsilon})$ constant-round iterations. Each iteration involves one witness technique application where each party reliably broadcasts its current value in $\mathbb{R}$ (with a reliable broadcast protocol such as Bracha's [5]), and obtains at least $n - t$ reliably broadcast values. The technique ensures that every two parties obtain the values of at least $n-t$ common parties. Since the technique's introduction in [1], most asynchronous approximate agreement protocols have depended on it. Some examples are [1, 12] for agreement in $\mathbb{R}$, [14] for agreement in $\mathbb{R}^d$ when $d \geq 2$ and [15] for agreement in graphs (trees, chordal graphs, cycle-free semilattices). Note that this list is not exhaustive; we mention some other examples in the full version [10].

The witness technique requires the parties to reliably broadcast their inputs. Since reliable broadcast requires $\Omega(n^2)$ messages for deterministic [9] or strongly adaptive [2] security against $t = \Omega(n)$ faults, the witness technique requires $\Omega(n^3)$ messages to be sent. Hence, the optimally resilient approximate agreement protocol of Abraham, Amit and Dolev [1] costs $\Theta(n^3)$ messages per iteration. This is the case despite asynchronous approximate agreement being possible with $\Theta(n^2)$ messages per iteration, as demonstrated by the protocol of Dolev et al. [8] which suboptimally tolerates $t < \frac{n}{5}$ faults. Therefore, we ask the following question: *Is there an asynchronous approximate agreement protocol that optimally tolerates $t < \frac{n}{3}$ faults with only a quadratic (proportional to n^2) amount of communication?*

In this work, we answer this question affirmatively by abandoning the witness technique. First, we achieve edge agreement (a discrete form of approximate agreement) in finite trees [15] with the optimal resilience $t < \frac{n}{3}$ via multivalued 2-graded consensus iterations. Then, we extend our protocol to achieve edge agreement in the infinite path $\mathbb{Z}$. Finally, we show that edge agreement in $\mathbb{Z}$ implies ε-agreement in $\mathbb{R}$ by reducing the latter to the former. Our final protocol for ε-agreement in $\mathbb{R}$ takes $6 \log_2 \frac{M}{\varepsilon} + \mathcal{O}(\log \log \frac{M}{\varepsilon})$ rounds (where M is the maximum honest input magnitude), with $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log \log \frac{M}{\varepsilon})$ sent per round.

Our work is inspired by [13], which achieves exact convex agreement in $\mathbb{Z}$ with byzantine agreement iterations in a synchronous network. We instead achieve edge agreement in $\mathbb{Z}$ with graded consensus, which is much simpler than byzantine agreement, especially in asynchronous networks. Note though that [13] supports large inputs with less communication than us.

2 Model & Definitions

We consider an asynchronous network of n message-sending parties $P_1, P_2, \ldots, P_n$, fully connected via reliable and authenticated channels. An adversary corrupts up to $t < \frac{n}{3}$ parties, making them byzantine, and these parties become controlled by the adversary. The adversary adaptively chooses the parties it wants to corrupt during protocol execution, depending on the messages sent over the network. If a party is never corrupted, then we call it honest.

The parties do not have synchronized clocks. The adversary can schedule messages as it sees fit, and it is only required to eventually deliver messages with honest senders. If a party sends a message, then the adversary may corrupt the party instead of delivering the message.

To define asynchronous round complexity, we imagine an external clock. If a protocol runs in R rounds, then the time elapsed between when every honest party running the protocol knows its input and when every honest party outputs/terminates is at most $R\Delta$, where Δ is the maximum honest message delay in the protocol's execution.

Edge Agreement in a Tree

In edge agreement in a tree graph $T = (V, E)$, each party P_i acquires an input vertex $v_i \in V$, and outputs a vertex $y_i \in V$. We want the following properties:

- ***edge agreement:*** Every two honest output vertices are either equal or adjacent in T.
- ***convex validity:*** For every honest output y, there exist some (possibly equal) honest inputs v_y and v'_y such that y is on the path which connects v_y and v'_y in T.

Edge agreement in a tree generalizes edge agreement in a path, which is essentially the same task as approximate agreement in an interval in $\mathbb{R}$, but with the input/output domain restricted to the integers (with adjacent integers representing adjacent path vertices).

Graded Consensus

In k-graded consensus, each party P_i acquires an input v_i in an input domain $\mathcal{M}$, and outputs some value-grade pair $(y_i, g_i) \in (\mathcal{M} \times \{1, \ldots, k\}) \cup \{(\perp, 0)\}$. The following must hold:

- ***agreement:*** If any honest parties P_i and P_j output (y_i, g_i) and (y_j, g_j), then $|g_i - g_j| \leq 1$, and if $\min(g_i, g_j) \geq 1$, then $y_i = y_j$.
- ***intrusion tolerance:*** If $(y, g) \neq (\perp, 0)$ is an honest output, then y is an honest input.
- ***validity:*** If the honest parties have a common input $m \in \mathcal{M}$, then they all output (m, k).

As [3] has noted before, k-graded consensus is equivalent to edge agreement in a particular tree when the inputs must all be leaves of the tree.

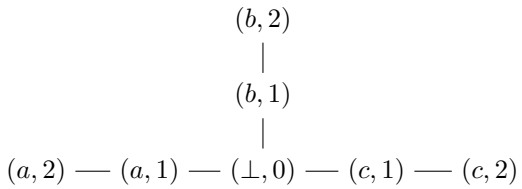


Figure 1 Edge agreement in a spider tree with the center $(\perp, 0)$ and a path $((m, 1), \ldots, (m, k))$ attached to it for each $m \in \mathcal{M}$ is equivalent to k-graded consensus when the parties can only have leaf edge agreement inputs, with each leaf input (m, k) in bijection with the k-graded consensus input m.

In the full version [10], we construct a family of multivalued 2^k-graded consensus protocols $\mathsf{GC}_{2^0}, \mathsf{GC}_{2^1}, \mathsf{GC}_{2^2}, \ldots$ which each take $3k + 3$ rounds, with $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log k + \log |\mathcal{M}|)$ per round. These are the most efficient multivalued graded consensus protocols we are aware of. In the full version we make use of the 6-round 2-graded consensus protocol GC_2, while in this brief announcement we present a simplified approximate agreement protocol that uses 3-graded consensus, which the parties can reach by running the 9-round 4-graded consensus protocol GC_4 and subtracting 1 from their output grades if they obtain the grade 4.

3 Overview & Contributions

Our first contribution is a protocol for edge agreement in finite trees. Against byzantine faults, this problem was first studied by Nowak and Rybicki [15]. It generalizes both edge agreement in finite paths (the discrete version of ε-agreement in $[0, 1]$) and graded consensus.

Nowak and Rybicki achieve edge agreement in a finite tree $T = (V, E)$ of diameter D with $\lceil \log_2 D \rceil + 1$ constant-round witness technique iterations and thus $\Theta(n^3 \log D(\log |V| + \log n))$ bits of communication, where the $\log n$ term is due to the party IDs that identify each reliable

broadcast's sender party. Meanwhile, we achieve edge agreement with at most $h(T)$ iterations ($6h(T)+1$ rounds), where $h(T)$ (T's centroid decomposition [16] height) is a property we define in the full version [10]. The integer value $h(T)$ can be anywhere in $[\lfloor \log_2 D \rfloor, \lceil \log_2 |V| \rceil]$, which means that our protocol's round complexity is for some trees (though not for spider trees, trees with $\mathcal{O}(D)$ vertices etc.) worse than Nowak and Rybicki's. However, our iterations only cost $\mathcal{O}(n^2)$ messages, each of size at most $\mathcal{O}(\log \Delta + \log(h(T)))$ where Δ is T's maximum degree. So, our protocol requires roughly n times less communication when $h(T) \approx \log_2 D$.

In the full version [10], we present a parametrized recursive protocol $\mathsf{TC}(T)$ that achieves edge agreement in any given finite tree T. On a high level, it works as follows:

1. If T has 1 or 2 vertices, then each party outputs its input vertex. This is the base case.
2. If T has $s \geq 3$ vertices, then the parties let σ be a centroid vertex of T (whose deletion from T results in a forest whose components all have at most $s/2$ vertices), and let $w_1, \ldots, w_d$ be σ's neighbors sorted by vertex index. Then, they run 2-graded consensus, where each party's input is either σ (if its edge agreement input is σ) or some neighbor w_k of σ (if its edge agreement input is in H_k, which is how we refer to the tree component of $T \setminus \{\sigma\}$ that contains w_k). If the parties reach consensus on σ, then they output σ. Otherwise, if they reach consensus on some neighbor w_k of σ, then the parties with input vertices outside H_k adopt the new input w_k, and we reduce the task to edge agreement in the subtree H_k.

There is a snag. The explanation above only works if the parties actually reach unanimous agreement on either σ or on one of its neighbors w_k. However, 2-graded consensus does not guarantee this, since some parties might output $(\perp, 0)$ from it. What allows us to overcome this issue is that if anybody outputs $(\perp, 0)$, then the parties all learn that they ran 2-graded consensus with differing inputs, and thus learn that σ is a safe output vertex w.r.t convex validity.

Our approach for finite trees above corresponds to binary search when the tree is a path. For example, the parties reach edge agreement in the path $(0, \ldots, 8)$ by either directly agreeing on 4, or by reducing the problem to edge agreement in either $(0, \ldots 3)$ or $(5, \ldots, 8)$. Binary search does not support the infinite path $\mathbb{Z}$. Fortunately, 2-graded consensus also enables exponential search. In the full version [10], we present a protocol for edge agreement in $\mathbb{Z}$ where we use 2-graded consensus to implement a strategy based on (doubly) exponential search.

When the maximum honest input magnitude is M, our protocol for edge agreement in $\mathbb{Z}$ takes $6 \log_2 M + \mathcal{O}(\log \log M)$ rounds, with $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log \log M)$ per round. In the full version [10], we reduce ε-agreement in $\mathbb{R}$ to edge agreement in $\mathbb{Z}$ to show that this implies ε-agreement in $\mathbb{R}$ in $6 \log_2 \frac{M}{\varepsilon} + \mathcal{O}(\log \log \frac{M}{\varepsilon})$ rounds with $\mathcal{O}(n^2 \log \frac{M}{\varepsilon})$ messages and $\mathcal{O}(n^2 \log \frac{M}{\varepsilon} \log \log \frac{M}{\varepsilon})$ bits of communication in total. Note that the factor 6 in the round complexity here stems from us using our 6-round 2-graded consensus protocol.

In terms of message and communication (though not round) complexity, our protocol for ε-agreement in $\mathbb{R}$ is more efficient than that of Abraham et al. [1], who achieve ε-agreement in $\mathbb{R}$ with $\mathcal{O}(\log \frac{S}{\varepsilon})$ constant-round witness technique iterations (where S is the honest input spread, i.e. the maximum difference of any honest inputs), and with $\Theta(n^3 \log \frac{S}{\varepsilon})$ messages in total.

Another notable protocol is Delphi, by Bandarupalli, Bhat, Bagchi, Kate, Liu-Zhang and Reiter [4]. To efficiently achieve ε-agreement with ℓ-bit inputs in $\mathbb{R}$, they assume an input distribution (normal distribution for the following), and when the honest input spread is S they achieve ε-agreement except with probability $2^{-\lambda}$ in $\mathcal{O}(\log(\frac{S}{\varepsilon} \log \frac{S}{\varepsilon}) + \log(\lambda \log n))$ rounds with $\mathcal{O}(\ell n^2 \frac{S}{\varepsilon}(\log(\frac{S}{\varepsilon} \log \frac{D}{\varepsilon}) + \log(\lambda \log n)))$ bits of communication, while relaxing validity by allowing outputs outside the range of the honest inputs by at most S. They use the security parameter λ here to assume bounds on S that hold except with $2^{-\lambda}$ probability thanks to their input distribution assumptions. In comparison, we achieve ε-agreement in $\mathbb{R}$ without relaxing validity or assuming any input bounds. As Table 1 shows, our protocol is also more efficient, in particular since Delphi requires a cubic amount of communication per round when $S \geq n \cdot \varepsilon$.

Table 1 Comparison of protocols for asynchronous ε-agreement in $\mathbb{R}$ when the parties have inputs in $[0,1]$. If v_{lo} and v_{hi} are the minimum and maximum honest inputs, then $S = v_{\mathsf{hi}} - v_{\mathsf{lo}}$ and $M = v_{\mathsf{hi}}$. To make the comparisons simple, we assume for [8], [1] and [4] that the inputs are all multiples of ε.

Threshold	Bits Sent / Round	Round Complexity	Relaxation[a]	Source
$t < \frac{n}{5}$	$\mathcal{O}(n^2 \log \frac{1}{\varepsilon})$	$\lceil \log_2 \frac{1}{\varepsilon} \rceil$	0	[8]
$t < \frac{n}{3}$	$\mathcal{O}(n^3 \log \frac{n}{\varepsilon})$[b]	$\mathcal{O}(\log \frac{S}{\varepsilon})$	0	[1]
$t < \frac{n}{3}$	$\mathcal{O}(n^2 \min(\frac{S}{\varepsilon}, n \log \frac{1}{\varepsilon}))$	$\mathcal{O}(\log(\frac{\log(1/\varepsilon)\min(1/\varepsilon,n)}{\varepsilon}))$	S	[4]
$t < \frac{n}{3}$	$\mathcal{O}(n^2 \log\log \frac{M}{\varepsilon})$[c]	$\mathcal{O}(\log \frac{M}{\varepsilon})$	0	this work

a) The relaxation is how far an honest output is allowed to be from the honest input range $[v_{\mathsf{lo}}, v_{\mathsf{hi}}]$.
b) The first few rounds of [1] estimate the spread S, and this costs $\Theta(n^4 \log \frac{1}{\varepsilon})$ bits of communication. However, this can be reduced to $\Theta(n^3 \log \frac{n}{\varepsilon})$ with modern reliable broadcast protocols [6].
c) The $\log\log \frac{M}{\varepsilon}$ factor here is for tags that distinguish messages sent in different protocol iterations.

4 Simplified Approximate Agreement in $[0,1]$

In this section, we present a simplified ε-agreement protocol that is better suited for a brief announcement than the more complicated full protocol in the full version [10]. While the full protocol uses 2-graded consensus, the simplified protocol in this section uses 3-graded consensus, which makes it less round-efficient since (with our graded consensus constructions) 3-graded consensus takes 9 rounds while 2-graded consensus takes 6. Moreover, the simplified protocol only supports inputs in bounded intervals rather than inputs in $\mathbb{R}$. Still, the protocols share the same core idea, which is that the parties repeatedly bisect the interval where their values reside by reaching graded consensus on whether their values are low or high.

In $\mathsf{Apx}(a,b)$, the parameterized approximate agreement protocol we present in this section, the parties reach ε-agreement in an interval $[a,b]$ by using 3-graded consensus to recursively reduce ε-agreement in $[a,b]$ to ε-agreement in either $[a, \frac{a+b}{2}]$ (reached via $\mathsf{Apx}(a, \frac{a+b}{2})$) or $[\frac{a+b}{2}, b]$ (reached via $\mathsf{Apx}(\frac{a+b}{2}, v)$). The parties reach 3-graded consensus on whether their inputs are below the midpoint $\frac{a+b}{2}$ or not. If they all have inputs below (resp. above) $\frac{a+b}{2}$, then they unanimously agree that this is the case, and so they run $\mathsf{Apx}(a, \frac{a+b}{2})$ (resp. $\mathsf{Apx}(\frac{a+b}{2}, b)$) to reach approximate agreement. Otherwise, $\frac{a+b}{2}$ is in the honest input range, and this fact lets us assign an appropriate behavior to each 3-graded consensus output so that the parties reach approximate agreement no matter which two 3-graded consensus outputs they settle on.

When the parties run $\mathsf{Apx}(0,1)$ for ε-agreement in $[0,1]$, they reach the base case of a recursive $\mathsf{Apx}(a,b)$ instance where $b-a \leq \varepsilon$ (where they can just output their inputs) with $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ recursive Apx calls, or in other words $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ iterations of 3-graded consensus. In total, this costs $9\lceil \log_2 \frac{1}{\varepsilon} \rceil$ rounds, $\mathcal{O}(n^2 \log \frac{1}{\varepsilon})$ messages and $\mathcal{O}(n^2 \log \frac{1}{\varepsilon} \log\log \frac{1}{\varepsilon})$ bits of communication. Here, we have a $\log\log \frac{1}{\varepsilon}$ factor because the parties have to be able to tell to which 3-graded consensus iteration each message belongs to, and this requires $\mathcal{O}(\log\log \frac{1}{\varepsilon})$-bit message tags.

Note that Apx does not allow the parties to terminate (stop sending messages) after they output, since some parties not sending the messages they are supposed to send could lead to some other parties never obtaining outputs. We address this shortcoming in the full version [10] with a simple constant-round quadratic-complexity termination procedure.

Protocol $\mathsf{Apx}(a, b)$

Code for a party P_i with the input v_i

if $b - a \leq \varepsilon$ **then**
  output v_i and do not run the rest of the protocol
run an instance of a 3-graded consensus protocol GC_3 with the other parties where your input is `LEFT` if $v_i < \frac{a+b}{2}$, and `RIGHT` if $v_i \geq \frac{a+b}{2}$
wait until you output some (k, g) from GC_3
if $g = 3$ **then**
  let $v_i^{\mathsf{next}} \leftarrow \min(v_i, \frac{a+b}{2})$ if $k = \texttt{LEFT}$, and let $v_i^{\mathsf{next}} \leftarrow \max(v_i, \frac{a+b}{2})$ if $k = \texttt{RIGHT}$
else
  $v_i^{\mathsf{next}} \leftarrow \frac{a+b}{2}$
if $g \leq 1$ **then**
  output $\frac{a+b}{2}$
if $k = \texttt{LEFT}$ **then**
  run an instance $\mathsf{Apx}(a, \frac{a+b}{2})$ with the other parties where your input is v_i^{next}
  when you output y from $\mathsf{Apx}(a, \frac{a+b}{2})$, output y from $\mathsf{Apx}(a, b)$ if $g \geq 2$
else if $k = \texttt{RIGHT}$ **then**
  run an instance $\mathsf{Apx}(\frac{a+b}{2}, b)$ with the other parties where your input is v_i^{next}
  when you output y from $\mathsf{Apx}(\frac{a+b}{2}, b)$, output y from $\mathsf{Apx}(a, b)$ if $g \geq 2$

If the parties all run $\mathsf{Apx}(a, b)$ with inputs in $[a, \frac{a+b}{2})$, then each party P_i runs GC_3 with the input `LEFT`, outputs $(\texttt{LEFT}, 3)$ from GC_3, and obtains its final output from an $\mathsf{Apx}(a, \frac{a+b}{2})$ instance (that everybody runs) where its input is $v_i^{\mathsf{next}} = v_i$. So, $\mathsf{Apx}(a, b)$'s security recursively follows from $\mathsf{Apx}(a, \frac{a+b}{2})$'s security. Likewise, if the parties all run $\mathsf{Apx}(a, b)$ with inputs in $[\frac{a+b}{2}, b]$, then they recursively reach ε-agreement via $\mathsf{Apx}(\frac{a+b}{2}, b)$.

On the other hand, if neither $[a, \frac{a+b}{2})$ nor $[\frac{a+b}{2}, b]$ contain all inputs, then $\frac{a+b}{2}$ is a safe output value w.r.t. validity. Knowing this, we can prove $\mathsf{Apx}(a, b)$'s security via case analysis.

- If the parties all output $(\texttt{LEFT}, 3)$ or $(\texttt{LEFT}, 2)$ from GC_3, then they all run $\mathsf{Apx}(a, \frac{a+b}{2})$ with inputs in $[a, \frac{a+b}{2}]$ and obtain their outputs from it. So, security follows from $\mathsf{Apx}(a, \frac{a+b}{2})$. Some parties (those with $\mathsf{Apx}(a, b)$ inputs above $\frac{a+b}{2}$ and those with the GC_3 grade 2) run $\mathsf{Apx}(a, \frac{a+b}{2})$ with the input $v_i^{\mathsf{next}} = \frac{a+b}{2}$ instead of v_i. This is fine because $\frac{a+b}{2}$ is safe.
- If the parties all output $(\texttt{LEFT}, 2)$ or $(\texttt{LEFT}, 1)$ from GC_3, then they all run $\mathsf{Apx}(a, \frac{a+b}{2})$ with the input $\frac{a+b}{2}$, and thus all output $\frac{a+b}{2}$ from it. The parties with the GC_3 grade 2 output $\frac{a+b}{2}$ from $\mathsf{Apx}(a, b)$ once they output this from $\mathsf{Apx}(a, \frac{a+b}{2})$, while the ones with the GC_3 grade 1 output $\frac{a+b}{2}$ from $\mathsf{Apx}(a, b)$ directly after they obtain the GC_3 grade 1.
- If the parties all output $(\texttt{LEFT}, 1)$ or $(\perp, 0)$ from GC_3, then they all directly output $\frac{a+b}{2}$ from $\mathsf{Apx}(a, b)$ after they output from GC_3. Here, it does not matter that the parties with the GC_3 grade 1 run $\mathsf{Apx}(a, \frac{a+b}{2})$ while the rest do not, because the parties with the GC_3 grade 1 do not care about their $\mathsf{Apx}(a, \frac{a+b}{2})$ outputs.
- The cases where some parties obtain the GC_3 value `RIGHT` are similar to the cases above.

Future Work. It remains open to design a protocol for ε-agreement in $\mathbb{R}$ that tolerates $t < \frac{n}{3}$ faults in $\mathcal{O}(\log \frac{S}{\varepsilon})$ rounds (where S is the honest input spread) with quadratic communication. We do not know of any such protocol for even synchronous networks, let alone asynchronous ones. The classical synchronous protocol in [8] which at first seems to fit the bill in fact takes as many rounds as the adversary desires because its round complexity scales with the spread of *all* inputs, including fake byzantine ones. It would be a good first step to solve this issue.

References

1 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, OPODIS '04, pages 229–239, Berlin, Heidelberg, 2004. Springer-Verlag. doi:10.1007/11516798_17.

2 Ittai Abraham, T-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 317–326, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331629.

3 Hagit Attiya and Jennifer L. Welch. Multi-Valued Connected Consensus: A New Perspective on Crusader Agreement and Adopt-Commit. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*, volume 286 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2023.6.

4 Akhil Bandarupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, Chen-Da Liu-Zhang, and Michael K. Reiter. Delphi: Efficient Asynchronous Approximate Agreement for Distributed Oracles. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 456–469, Los Alamitos, CA, USA, June 2024. IEEE Computer Society. doi:10.1109/DSN58291.2024.00051.

5 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2), 1987. doi:10.1016/0890-5401(87)90054-X.

6 Jinyuan Chen. Ociorcool: Faster byzantine agreement and reliable broadcast, 2024. doi:10.48550/arXiv.2409.06008.

7 D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

8 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, May 1986. doi:10.1145/5925.5931.

9 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, January 1985. doi:10.1145/2455.214112.

10 Mose Mizrahi Erbes and Roger Wattenhofer. Asynchronous approximate agreement with quadratic communication, 2025. doi:10.48550/arXiv.2408.05495.

11 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985. doi:10.1145/3149.214121.

12 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC '22, pages 70–80, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538442.

13 Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Communication-optimal convex agreement. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '25, pages 39–49, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3732772.3733551.

14 Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in byzantine asynchronous systems. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 391–400, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488657.

15 Thomas Nowak and Joel Rybicki. Byzantine Approximate Agreement on Graphs. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2019.29.

16 A simple introduction to centroid decomposition. A Simple Blog, 2020. Accessed: 2025-08-15. URL: https://robert1003.github.io/2020/01/16/centroid-decomposition.html.

Brief Announcement: Time, Fences and the Ordering of Events in TSO

Raïssa Nataf
Technion, Haifa, Israel

Yoram Moses
Technion, Haifa, Israel

Abstract

Total Store Order (TSO) is one of the most popular relaxed memory model in multiprocessor architectures, widely implemented, for example, in Intel's x86 and x64 platforms. It delays write visibility via store buffers, thereby allowing a significant improvement in efficiency. This, however, complicates reasoning about correctness, as executions may violate sequential consistency. We present a semantic framework that provides effective tools that can pinpoint when such synchronization is necessary under TSO. We define a TSO-specific *occurs-before* relation, adapting Lamport's happens-before to TSO, and prove that events at different sites can be temporally ordered only via an occurs-before chain. Analyzing how fences and `RMW`s create these chains lets us identify when they are unavoidable. We present in this BA how these results impact linearizable implementations of registers, capturing information flow and causality in TSO. The full version of this work provides details as well as results regarding the need for synchronization in linearizable implementations of additional objects.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Concurrent algorithms

Keywords and phrases TSO, linearizability, happens before, fences, synchronization actions

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.62

Related Version *Full Version*: `https://arxiv.org/abs/2508.11415` [15]

1 Introduction and Related Work

Modern multiprocessors rely on relaxed memory models to improve performance through techniques such as store buffering and out-of-order execution. Among these, *Total Store Order (TSO)* – used in Intel's x86 and x64 architectures – is one of the most widely deployed. TSO increases efficiency by allowing writes to be held temporarily in per-process store buffers, deferring their visibility to other processors. However, this optimization breaks *sequential consistency* (SC) [11], making reasoning about correctness more difficult: a program correct under SC may fail under TSO due to delayed write visibility. To ensure correct behavior under TSO, programmers can use synchronization primitives such as *memory fences* (`F`) and *read-modify-write* (`RMW`) operations. These enforce memory ordering by flushing buffers or performing atomic accesses. While effective, they limit concurrency and reduce performance [4, 9, 17]. Determining *when such synchronization is truly necessary* is therefore a central question in the study of weak memory models such as TSO.

Lamport's *happens-before* relation [10] is central in asynchronous computing, underlying vector clocks, race detection, causal memory [12, 7, 1], and more. Happens-before was originally defined in asynchronous message-passing models, and it captures all of the information about timing that processes can obtain in such settings. In a recent paper, we proved a theorem called Delaying the Future (DtF) that provides a close formal connection between happens-before and the ability to reorder actions and events in asynchronous message-passing systems [14]. Roughly speaking, DtF implies that if there is no message chain between an


© Raïssa Nataf and Yoram Moses;
licensed under Creative Commons License CC-BY 4.0



operation A and an operation B in a given run r, then there is a run r' that is indistinguishable to all processes from r in which A takes place after B does in real-time. This is of importance, for example, when considering linearizable implementations of concurrent objects, since the real-time order of event invocations and completions plays a central role in the definition of Linearizability [8]. The happens-before relation has been considered in different models, including TSO [3]. In this work, we introduce a new definition for the analogue of happens-before in TSO, which we call the *occurs-before* relation. One of the differences is that our definition stems from an operational model of TSO, whereas previous definitions such as in [3] are formulated using declarative models. As a result, our approach employs a lower level of abstraction, making it easier to derive concrete implementation constraints. We demonstrate the analogy between the classical happens-before relation of message chains to our occurs-before relation by proving a version of the DtF theorem for TSO, which is obtained by replacing "happens before" by "occurs before." This makes it possible to prove necessary conditions on the use of synchronization in linearizable TSO implementations, much in the spirit and style that [14] does for linearizable register implementations in asynchronous message passing.

Related Work. Prior work has tackled the need for synchronization actions in weak memory models from both practical and theoretical perspectives. Automated tools attempt to minimize fences while preserving correctness [2, 17], and impossibility results show that synchronization is sometimes unavoidable.

Attiya et al. [4] and Castañeda et al. [6] are most directly related to our TSO results. Both works establish conditions on when linearizable implementations require the use of synchronization operations. In [4], covering arguments are used to show that precise communication patterns involving reads and writes are unavoidable in implementing classic and widely used specifications. Their results apply to objects whose methods are strongly non-commutative – such as sets, queues, or stacks – but not to registers. The more recent [6] proves a mergeability theorem for TSO and related weak memory models traces and applies it to obtain results about objects with one-sided non commutative methods such as registers. They show that *linearizable* TSO implementations of objects, including registers, must use fences or `RMW`s in some executions. In [14], a DtF theorem is proved for asynchronous message-passing systems. It is then used to prove the need to construct message chains between operations in linearizable register implementations in that model.

The current paper can be viewed as performing an analogous analysis for TSO to that of [14]. Our analysis highlights the role of synchronization primitives, extending lower bounds on the use of synchronization primitives in TSO. Practically, such lower bounds identify the synchronization mechanisms implementations must employ and indicate when further attempts to remove them would be futile.

2 Model and Preliminary Definitions

This section outlines the main features of our model; the reader should consult the full version [15] for additional details. The model is based on the operational definition of TSO given in [6]. (Similar, though slightly different, operational models for TSO have appeared, e.g., in [16, 5].) It consists of a set $\Pi = \{1, \ldots, n\}$ of n processes and a finite set Var of variables. A basic TSO state is a pair $\sigma = \langle \mathsf{m}, \mathsf{buf} \rangle$, where $\mathsf{m} \in \mathsf{Var} \to \mathsf{Val}$ describes the main memory and $\mathsf{buf} \in \Pi \to (\mathsf{Var} \times \mathsf{Val})^*$ assigns a queue called a *store buffer* to every process. Processes can perform actions from the set $\{\mathtt{R}[x], \mathtt{W}[x, v], \mathtt{F}, \mathtt{RMW}[x, v_{\mathsf{exp}}, v_{\mathsf{new}}], \bot\}$. These correspond,

respectively, to reading the value of x (from local buffer if it contains an unexecuted write to x and from the physical variable x in memory otherwise), writing the value v in x (v is appended to the local buffer), performing a *fence* (which occurs only when the local buffer is empty), performing a read-modify-write action $\texttt{RMW}[x, v_{\mathsf{exp}}, v_{\mathsf{new}}]$ (which occurs only if the buffer is empty and the value of x in the memory is v_{exp}. Its effect is to write v_{new} directly to the memory). To account for propagating values from the store buffer to variables in memory, we associate with each $i \in \Pi$ a "local dispatcher" component d_i with a single action `prop`, whose impact is to propagate a write command from the buffer queue to the memory. We denote $\mathrm{Ag} \triangleq \Pi \cup \{d_i\}_{i\in\Pi}$ and call its elements *agents*. Reads and writes are always enabled, while fences, `RMW` and `prop` actions can take place only if specific preconditions hold.

We think of an action taking place as being an *event*. It is convenient to consider two types of read events: $\texttt{RfB}(x, v)$ and $\texttt{RfM}(x, v)$. The former is when the value of x is read from the store buffer, and the latter is when the value of x is read from the variable $x \in \mathsf{m}$ in memory. In both instances, the value returned by the read action is $v \in \mathsf{Val}$. Using $\beta|_x$ to denote the restriction of a store buffer β to pairs of the form $\langle x, _\rangle$ involving writes to $x \in \mathsf{m}$, the preconditions and effects of the event of a single action being performed in a TSO state $\sigma = \langle \mathsf{m}, \mathsf{buf}\rangle$ are defined as follows:

WRITE
$$\frac{e = i : \texttt{W}(x, v) \qquad \mathsf{buf}' = \mathsf{buf}[i \mapsto \mathsf{buf}(i) \cdot \langle x, v\rangle]}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}, \mathsf{buf}'\rangle}$$

READ-FROM-BUFFER
$$\frac{e = i : \texttt{RfB}(x, v) \qquad \mathsf{buf}(i)|_x = _ \cdot \langle\langle x, v\rangle\rangle}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}, \mathsf{buf}\rangle}$$

READ-FROM-MEMORY
$$\frac{e = i : \texttt{RfM}(x, v) \qquad \mathsf{buf}(i)|_x = \varepsilon \qquad \mathsf{m}(x) = v}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}, \mathsf{buf}\rangle}$$

RMW
$$\frac{e = i : \texttt{RMW}(x, v_{\mathsf{exp}}, v_{\mathsf{new}}) \qquad \mathsf{buf}(i) = \varepsilon \qquad \mathsf{m}(x) = v_{\mathsf{exp}}}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}[x \mapsto v_{\mathsf{new}}], \mathsf{buf}\rangle}$$

FENCE
$$\frac{e = i : \texttt{F} \qquad \mathsf{buf}(i) = \varepsilon}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}, \mathsf{buf}\rangle}$$

PROPAGATE
$$\frac{e = d_i : \texttt{prop}(x, v) \qquad \mathsf{buf}(i) = \langle\langle x, v\rangle\rangle \cdot \beta \qquad \mathsf{m}' = \mathsf{m}[x \mapsto v] \qquad b' = b[i \mapsto \beta]}{\langle \mathsf{m}, \mathsf{buf}\rangle \xrightarrow{e} \langle \mathsf{m}', \mathsf{buf}'\rangle}$$

A TSO action is considered a *memory access* of the variable x if it is either a $\texttt{RMW}[x, \cdot]$, a `prop` action resulting in $\texttt{prop}(x, \cdot)$, or a read action $\texttt{R}[x]$ resulting in $\texttt{RfM}(x, \cdot)$.

Runs and Protocols. A protocol $P = (P_1, \ldots, P_n)$ maps each process' local state to a nonempty set of enabled actions. Local states record the sequence of actions observed so far. A global state is $(\sigma, \ell_1, \ldots, \ell_n)$ with σ being the TSO state and ℓ_i the local state of process i. A *run* is an infinite sequence of global states, starting with empty buffers and initial local states; each step applies a *joint action* of process, dispatcher, and environment actions, with no conflicting memory accesses to the same variable. A run is of P if every process action in it is allowed by P. We make two liveness assumptions: (i) every enabled `prop` eventually occurs, ensuring fences terminate, and (ii) any process with an enabled action eventually moves.

Nodes and Tags. A node $\theta = \langle b, t\rangle$ denotes agent b at time t; $\theta.\alpha$ is b's action at time t (possibly a null action). The k^{th} write $\texttt{W}(x, v)$ by i is taken to have a *tag* $\texttt{W}.tag = \langle i, k\rangle$; the matching `prop`, `RfB`, and `RfM` on that value inherit the same tag.

3 The Occurs-before Relation in TSO

In asynchronous message-passing systems, the only way that a protocol can ensure that an action takes place later than a specific event at another process, is by forcing the action to be delayed until a message chain from the second process has been constructed. This

is Lamport's *happens-before* relation ([10]). We now define an analogous relation for TSO, where scheduling and propagation are asynchronous and processes lack direct knowledge of event timing. Complicating matters is the fact that distinct processes can read different values of a variable x *at the same time*. Our *occurs-before* relation, also builds chains across process timelines,[1] and is defined as follows:

▶ **Definition 1** (TSO occurs before). *Let $t < t'$, let $i, j \in \Pi$ and let $b, c \in \mathrm{Ag}$. We define the binary* occurs before *relation $\overset{ob}{\rightsquigarrow}_r$ between nodes in r as follows:*

1. $\langle b,t\rangle \overset{ob}{\rightsquigarrow}_r \langle b,t'\rangle$ *for every agent b.*
2. $\langle i,t\rangle \overset{ob}{\rightsquigarrow}_r \langle d_i,t'\rangle$ *if* $\langle i,t\rangle.\alpha \in \{\mathtt{W},\mathtt{RfB}\}$, $\langle d_i,t'\rangle.\alpha = \mathtt{prop}$ *and* $\langle i,t\rangle.\alpha.tag = \langle d_i,t'\rangle.\alpha.tag$.
3. $\langle b,t\rangle \overset{ob}{\rightsquigarrow}_r \langle c,t'\rangle$ *if $\langle b,t\rangle.\alpha$ and $\langle c,t'\rangle.\alpha$ are both memory access of the same variable* ***unless***
   a. $\langle b,t\rangle.\alpha$ *and* $\langle c,t'\rangle.\alpha$ *are both* $\mathtt{RfM}$ *actions, or*
   b. $b = d_i$, $c = i$, $\langle d_i,t\rangle.\alpha = \mathtt{prop}$ *and* $\langle i,t'\rangle.\alpha = \mathtt{RfM}$.
4. $\langle d_i,t\rangle \overset{ob}{\rightsquigarrow}_r \langle i,t'\rangle$ *if* $\langle d_i,t\rangle.\alpha = \mathtt{prop}$ *and* $\langle i,t'\rangle.\alpha \in \{\mathtt{F},\mathtt{RMW}\}$.
5. $\theta \overset{ob}{\rightsquigarrow}_r \theta'$ *if* $\theta \overset{ob}{\rightsquigarrow}_r \hat{\theta}$ *and* $\hat{\theta} \overset{ob}{\rightsquigarrow}_r \theta'$ *for some node $\hat{\theta}$.*

What makes $\overset{ob}{\rightsquigarrow}_r$ interesting and useful is the fact that, in a precise sense, it covers all the information that may be available to the processes regarding the ordering of events. We will show that, roughly speaking, if an event e in r is not related by $\overset{ob}{\rightsquigarrow}_r$ to another event e' in r, then there is a run r' that is indistinguishable from r in which e' takes place strictly before e does. We now turn to prove a more general result which will imply this fact.

4 Delaying the Future in TSO and its implications

Our goal is to use the occurs-before relation to prove lower bounds and necessary conditions on protocol structure. First, we show that processes cannot guarantee a specific event order without an occurs-before chain. If a process has the same local state at two execution points where it moves, it will take the same action at both points; events that did not modify its state cannot affect its behavior. This motivates the definitions of $\mathtt{Past}_r(S)$, the set of nodes in the causal past of S and of runs *local equivalence*.

▶ **Definition 2** (The past).[2] *For a set of nodes S in a run r, we define* $\mathtt{Past}_r(S) \triangleq \{\theta : \theta \overset{ob}{\rightsquigarrow}_r \theta' \text{ for some } \theta' \in S\}$ *and* $\mathtt{Past}^+_r(S) \triangleq \mathtt{Past}_r(S) \cup S$.

▶ **Definition 3** (Local Equivalence). *Two runs r and r' are called* locally equivalent, *denoted by $r \approx r'$, if for every process j, a local state ℓ_j of j appears in r iff ℓ_j appears in r'.*

We can now show:

▶ **Theorem 4** (Delaying the future in TSO). *Let r be a TSO run of a protocol P, let S be a set of nodes of r, and let $\Delta \geq 0$. Then there exists a TSO run $r' \approx r$ of the same protocol P such that:*

(a) *Every agent $b \in \mathrm{Ag}$ performs exactly the same actions in the same order in both runs; moreover, an action that b performs at time t in r is performed in r' at time t if $\langle b,t\rangle \in \mathtt{Past}_r(S)$, and it is performed at time $t + \Delta$ otherwise.*

[1] We do not call this relation "happens-before" because the latter has become synonymous with message chains over the last five decades, and the new relation in TSO has a different flavor.
[2] See Figure 1 for an illustration.

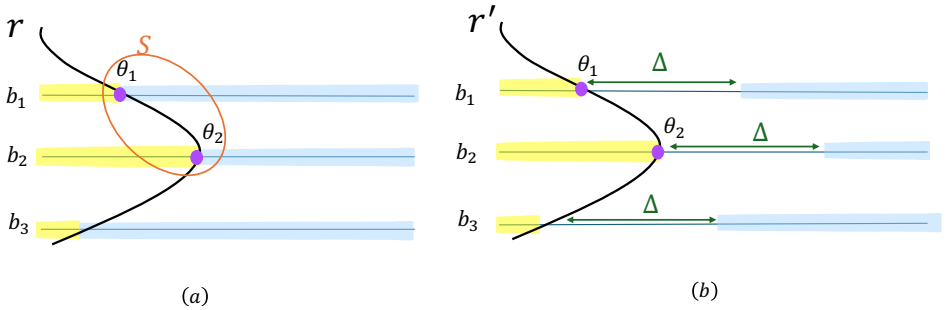


Figure 1 $S = \{\theta_1, \theta_2\}$ and $\texttt{Past}_r(S)$ is composed of all the nodes in the yellow part. $\texttt{Past}_r^+(S)$ contains in addition the nodes on the black curve, delimiting the nodes in past of S from the other nodes. (b) represents the run r' guaranteed to exist by Theorem 4.

(b) *Every process j's local states are shifted accordingly:*

$$r_j(t) = \begin{cases} {r'}_j(t) & \text{for all } \langle j, t\rangle \in \texttt{Past}_r^+(S) \\ {r'}_j(t+\Delta) & \text{otherwise.} \end{cases}$$

Intuitively, Theorem 4 implies that everything outside the $\texttt{Past}_r$ of a node (or set of nodes) can be moved forward in time by an arbitrary amount, while preserving the timing of events in this past. This implies that without an $\overset{ob}{\rightsquigarrow}$ connection, events and operations preceding a given operation can be shifted to no longer precede it, without changing its outcome. If changing the real-time order of two operations can affect the correctness of a given execution, as it typically does when linearizability is required, then implementations must ensure that operations whose order should not be changed are related by $\overset{ob}{\rightsquigarrow}$. In some cases, e.g., when a process runs solo, it is possible to show that establishing $\overset{ob}{\rightsquigarrow}$ requires a process to perform explicit synchronization operations. This is the subject of the next section.

5 Implementing Linearizable Operations in TSO

Roughly speaking, an implementation $\mathcal{I}$ of an object is a protocol satisfying the object's specification. It is said to be *linearizable* if in every execution of $\mathcal{I}$, operations appear to occur instantaneously in a way that is consistent with the original execution and that satisfy the sequential specification of the object. See [8] for a formal definition.

Registers. We consider implementations of a multi-writer multi-reader register, where every process may perform (linearizable) Read and Write operations, in the TSO memory model. For obstruction-free register implementations, techniques used in the proof of Theorem 4 and in [13] can be used to show:

▶ **Corollary 5.** *Let $n \geq 2$ and let $\mathcal{I}$ be an obstruction-free linearizable implementation of a register in TSO for n processes. Then for every $m > 0$ there must be a run r_m of $\mathcal{I}$ in which exactly m Write operations are performed, and each of the* Write*s performs a fence* F *or an* RMW*.*

We remark that Castañeda *et al.* prove in [6] that *spec-available* implementations of linearizable registers in TSO must contain a run in which at least one Read or Write operation performs an F or RMW action. Since every obstruction-free implementation is in particular

spec-available, the same follows for obstruction-free implementations. For obstruction-free implementations, Corollary 5 provides sharper bounds. In [15] Theorem 4 (DtF) is applied in the style of Theorem 6 of [14] to show that linearizable register implementations must create $\overset{ob}{\rightsquigarrow}$ relations between Write operations, to ensure linearizability. In addition, DtF is applied to the analysis of linearizable implementations of snapshot objects.

The Need for Synchronization to Establish Occurs Before. In a precise sense, Corollary 5 makes use of the fact that a process performing a linearizable Write must guarantee that the value it is writing is recorded in memory. We can show that the only way for a process to know that the value it wrote can be read by others is either by way of a read that provides it with feedback from other processes or by synchronization.

▶ **Definition 6** (feedback loop). *We say that there is a* feedback loop *between nodes* $\langle i, t_1\rangle$ *and* $\langle i, t_2\rangle$ *in run* r *(and write* $\langle i, t_1\rangle \circlearrowleft_r \langle i, t_2\rangle$*) if there is a node* $\langle b, t\rangle$ *with* $b \notin \{i, d_i\}$ *such that* $\langle i, t_1\rangle \overset{ob}{\rightsquigarrow}_r \langle b, t\rangle \overset{ob}{\rightsquigarrow}_r \langle i, t_2\rangle$*. We say that an operation* X *contains a feedback loop in* r *if* X.$s \circlearrowleft_r$ X.e.

▶ **Theorem 7.** *Let* X *be an operation in* r *containing a write with a tag* $\kappa = \langle i, k\rangle$*. If*

1. X *does not contain a feedback loop, and*
2. i *does not perform a* `F` *or* `RMW` *action during* X.

Then there is a run $r' \approx r$ *in which* κ *is not propagated to memory before* X.e.

This is a powerful result. In TSO, a process shares information only by writing to memory and making the write visible. If completing an operation requires others to observe it, the process must either obtain feedback from another process or use a `F` or `RMW` operation. In obstruction-free protocols, feedback can not be guaranteed, so fences or `RMW` become necessary to ensure visibility before operations can be completed.

6 Discussion

While we focus in this BA on registers, the full version [15] also covers snapshots, and extending the approach to the study of linearizable implementations of other shared objects appears promising. The register results of [6] exploit the fact that Read operations are one-sided non-commutative with respect to Write operations, yielding (roughly) that there exist runs where at least one of two adjacent write–read operations must contain a `F` or `RMW`. Our framework strengthens this: we prove that there exist runs where in fact Write s must contain a `F` or `RMW`. In typical register implementations, Writes complete with an acknowledgment independently of preceding Writes, meaning they are not one-sided non-commutative. Nevertheless, our tools show that there are runs where Writes must contain `F` or `RMW`. In addition, the DtF theorem and its uses, both in the TSO setting and in [14], raise the question of whether similar results hold in other memory models. Finally, we observe that although our $\overset{ob}{\rightsquigarrow}$ relation in TSO is analogous to happens-before in message passing, it differs in that $\overset{ob}{\rightsquigarrow}$ does not necessarily convey information, even under full information, whereas happens-before does. Nevertheless, like happens-before, the $\overset{ob}{\rightsquigarrow}$ relation remains a necessary condition for ordering in many cases. This fundamental difference is both interesting and worth deeper investigation.

References

1 Mustaque Ahamad, Gil Neiger, James E Burns, Prince Kohli, and Phillip W Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9(1):37–49, 1995. doi:10.1007/BF01784241.

2 Jade Alglave, Daniel Kroening, Vincent Nimal, and Daniel Poetzl. Don't sit on the fence: A static analysis approach to automatic fence insertion. *ACM Trans. Program. Lang. Syst.*, 39(2), May 2017. doi:10.1145/2994593.

3 Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. Fences in weak memory models. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 258–272. Springer, 2010. doi:10.1007/978-3-642-14295-6_25.

4 Hagit Attiya, Rachid Guerraoui, Danny Hendler, Petr Kuznetsov, Maged M. Michael, and Martin Vechev. Laws of order: expensive synchronization in concurrent algorithms cannot be eliminated. In *38th POPL*, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1926385.1926442.

5 Gérard Boudol and Gustavo Petri. Relaxed memory models: an operational approach. In *36th POPL*, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1480881.1480930.

6 Armando Castañeda, Gregory V. Chockler, Brijesh Dongol, and Ori Lahav. What cannot be implemented on weak memory? In *DISC 2024*, LIPIcs, 2024. doi:10.4230/LIPICS.DISC.2024.11.

7 Cormac Flanagan and Stephen N Freund. Fasttrack: Efficient and precise dynamic race detection. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 121–133. ACM, 2009. doi:10.1145/1542476.1542490.

8 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, july 1990. doi:10.1145/78969.78972.

9 Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. Repairing sequential consistency in C/C++11. *SIGPLAN Not.*, 52(6):618–632, June 2017. doi:10.1145/3140587.3062352.

10 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, july 1978. doi:10.1145/359545.359563.

11 Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers c-28*, 9:690–691, 1979. doi:10.1109/TC.1979.1675439.

12 Friedemann Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

13 Yoram Moses. Relating knowledge and coordinated action: The knowledge of preconditions principle. In R. Ramanujam, editor, *15th TARK*, pages 231–245, 2015. doi:10.4204/EPTCS.215.17.

14 Raïssa Nataf and Yoram Moses. Communication Requirements for Linearizable Registers. In *DISC 2024*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2024. doi:10.4230/LIPIcs.DISC.2024.33.

15 Raïssa Nataf and Yoram Moses. Time, fences and the ordering of events in TSO, 2025. arXiv:2508.11415.

16 Scott Owens, Susmit Sarkar, and Peter Sewell. A better x86 memory model: x86-TSO. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 391–407, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-03359-9_27.

17 Viktor Vafeiadis and Francesco Zappa Nardelli. Verifying fence elimination optimisations. In *Proceedings of the 18th International Conference on Static Analysis*, SAS'11, pages 146–162, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-23702-7_14.

Brief Announcement: Optimal Dispersion Under Asynchrony

Debasish Pattanayak
Department of Computer Science and Engineering, Indian Institute of Technology Indore, India

Ajay D. Kshemkalyani
Department of Computer Science, University of Illinois Chicago, IL, USA

Manish Kumar
Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India

Anisur Rahaman Molla
Indian Statistical Institute, Kolkata, India

Gokarna Sharma
Department of Computer Science, Kent State University, OH, USA

Abstract

We study the dispersion problem in anonymous port-labeled graphs: $k \leq n$ mobile agents, each with a unique ID and initially located arbitrarily on the nodes of an n-node graph with maximum degree Δ, must autonomously relocate so that no node hosts more than one agent. Dispersion serves as a fundamental task in the distributed computing of mobile agents, and its complexity stems from key challenges in local coordination under anonymity and limited memory.

The goal is to minimize both the time to achieve dispersion and the memory required per agent. It is known that any algorithm requires $\Omega(k)$ time in the worst case, and $\Omega(\log k)$ bits of memory per agent. A recent result [9] gives an optimal $O(k)$-time algorithm in the synchronous setting and an $O(k \log k)$-time algorithm in the asynchronous setting, both using $O(\log(k + \Delta))$ bits. We close the complexity gap in the asynchronous setting by presenting the first dispersion algorithm that runs in optimal $O(k)$ time using $O(\log(k + \Delta))$ bits of memory per agent. Our solution relies on a novel technique for constructing a port-one tree in anonymous graphs, which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed algorithms, mobile agents, local communication, dispersion, asynchrony, port-one tree, time and memory complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.63

Related Version *Full Version*: `https://arxiv.org/abs/2507.01298` [16]

Funding *Manish Kumar*: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.

1 Introduction

The *dispersion* problem, denoted as DISPERSION, involves $k \leq n$ mobile agents initially placed arbitrarily on the nodes of an n-node anonymous graph with maximum degree Δ. The agents must autonomously relocate so that each occupies a distinct node. The objective is to design algorithms that optimize both time and memory complexities. Time complexity is the total time to achieve dispersion, while memory complexity is the maximum bits stored per agent. Fundamental limits exist: certain topologies require $\Omega(k)$ time, and $\Omega(\log k)$ memory bits per agent are necessary for unique identifiers [2].



© Debasish Pattanayak, Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma;
licensed under Creative Commons License CC-BY 4.0



The dispersion problem has been extensively studied, both in arbitrary undirected graphs [2, 7, 8, 9, 10, 12, 13, 18], directed graphs [6] and in graphs with restricted topologies such as trees [2], grids [3, 4, 11], and dynamic rings [1, 15]. Several works have also examined fault-tolerant variants of the problem [5, 14, 15, 17]. Focusing on the arbitrary undirected graph, the state-of-the-art results of Kshemkalyani et al. [9], presented an optimal $O(k)$-round synchronous ($\mathcal{SYNC}$) algorithm and an $O(k \log k)$-epoch asynchronous ($\mathcal{ASYNC}$) algorithm, both using only $O(\log(k + \Delta))$ bits per agent. Here, an epoch denotes the minimal time interval in which each agent completes at least one computation cycle, and is equivalent to a round in $\mathcal{SYNC}$. Motivated by this line of work, we address the central open question: *Can an optimal $O(k)$-epoch solution be designed for* DISPERSION *in the $\mathcal{ASYNC}$ setting?*

Contributions. We answer this question affirmatively by providing an optimal $O(k)$-epoch solution in $\mathcal{ASYNC}$ with $O(\log(k+\Delta))$ bits per agent, showing that synchrony is not essential for time-optimal dispersion. This is achieved through a novel construction of a *Port-One tree* (P1TREE), which prioritizes edges connected to a port labeled '1' at either endpoint. This prioritization allows agents to find empty neighbor nodes in $O(1)$ epochs, even in $\mathcal{ASYNC}$, a task that previously required $O(\log k)$ epochs. Since settling k agents requires visiting k empty nodes, and each visit takes $O(1)$ epochs, the entire algorithm achieves an optimal $O(k)$ time complexity.

Challenges. The primary obstacle blocking the path to optimal dispersion in $\mathcal{ASYNC}$ is finding empty neighboring nodes in $O(1)$-epochs. The state-of-the-art technique relied on an "oscillation" mechanism where agents made periodic trips to distinguish empty nodes from temporarily unoccupied ("vacated") nodes. This timing-based coordination fundamentally fails in $\mathcal{ASYNC}$ because agents lack a shared sense of time. This creates a barrier to achieving $O(k)$ epoch solutions analogous to $\mathcal{SYNC}$ using a DFS-based traversal strategy.

Our key contribution is to bypass this time-based coordination strategy with a structural approach. We introduce the Port-One Tree (P1TREE), a structure that allows for agents to distinguish between "vacated" and "empty" nodes by visiting port-one neighbors. Further, we overcome several non-trivial challenges to make this technique viable:

1. **Building a** P1TREE**:** We design a modified DFS that prioritizes "port-1" edges while avoiding cycles.
2. **Selection of "vacated" nodes:** We devise a selection rule that guarantees at least $\lfloor l/3 \rfloor$ vacant nodes in a tree of size l, providing enough scouts for probing.
3. **Keeping track of vacated nodes:** Information about a vacated node and its port 1 neighbor is stored on an agent (that had originally settled there), which travels with the DFS head, avoiding the high memory cost of storing it at neighbors.
4. **Parallel probe with vacated nodes:** Our probing protocol involves a multi-hop check (up to 3 hops). The properties of our vacant node selection guarantee that a scout can distinguish an empty node from a vacated one by checking for the presence of settled agents at its port 1 neighbor or in the scout pool. This check completes in $O(1)$ epochs.
5. **Returning scouts home:** After dispersion is achieved, scouts re-traverse the constructed P1TREE in a post-order fashion to return to their originally assigned nodes, a process we show takes $O(k)$ epochs.

For general initial configurations with multiple starting locations, we extend the size-based merging technique from prior work [13]. When two DFS explorations meet, the one with fewer settled agents is "absorbed" by the larger one, ensuring that a single, monotonically growing tree eventually covers all k agents within the $O(k)$ time bound.

2 Model and Preliminaries

Graph. We consider a simple, undirected, connected graph $G = (V, E)$, where $n = |V|$ and $m = |E|$. The graph nodes are *anonymous* (lack unique identifiers) but are *port-labeled*: at each node v, incident edges have distinct local labels from 1 to δ_v. An edge $\{u, v\}$ is associated with two port numbers, p_{uv} at u and p_{vu} at v, which are assigned independently. Nodes are memoryless.

Edge and Node Terminology. We encode an edge $\{u, v\}$ as a 4-tuple $e = [u, p_{uv}, p_{vu}, v]$. We define the type of an edge $\{u, v\}$ by $\texttt{type}(\{u, v\})$ based on its port numbers:

- `t11`, if $p_{uv} = 1$ and $p_{vu} = 1$.
- `tp1`, if $p_{uv} \neq 1$ and $p_{vu} = 1$.
- `t1q`, if $p_{uv} = 1$ and $p_{vu} \neq 1$.
- `tpq`, if $p_{uv} \neq 1$ and $p_{vu} \neq 1$.

Agents. The system has $k \leq n$ mobile agents, $A = \{a_1, \ldots, a_k\}$, each with a unique ID, a_i.ID. Agents are initially located arbitrarily. Communication is local: an agent at a node can only interact with other co-located agents.

Time and Complexity. Agents operate in asynchronous "Communicate-Compute-Move" (CCM) cycles. An *epoch* is a minimal time interval where each agent completes at least one CCM cycle. A subsequent epoch begins after the end of the current one. Two epochs may have different lengths of time. Time complexity is measured in epochs, and memory complexity is the maximum bits stored in an agent's persistent memory.

3 Port-One Tree and its Construction

In this section, we define the Port-One Tree (P1Tree), prove its existence, and describe a distributed construction method that forms the basis of our dispersion algorithm.

Port-One Tree (P1Tree). Intuitively, every vertex in a P1Tree is incident to at least one tree edge connected to a port 1 at one of its endpoints.

▶ **Definition 1** (Port-One Tree (P1Tree)). *Let $G = (V, E)$ be an anonymous port-labeled graph. A tree $\mathcal{T} \subseteq E$ is a* P1Tree *if each vertex $v \in \mathcal{T}$ has at least one incident edge $\{v, w\} \in \mathcal{T}$ such that $\texttt{type}(\{v, w\}) \in \{$`tp1`, `t11`, `t1q`$\}$.*

A P1Tree may not contain all edges of the eligible types, as including them might create cycles. Figure 1 shows an example. Since every node has a port labeled 1, it has an edge of type `t11` *or* `t1q`.

▶ **Lemma 2.** *For any port-labeled graph G, there exists at least one* P1Tree *$\mathcal{T}$.*

3.1 DFS-based Construction

We now describe a modified DFS traversal, `DFS_P1Tree()`, that constructs a P1Tree. This algorithm forms the backbone of our agent-based dispersion strategy. Each node has two states: EMPTY and OCCUPIED. We further categorize each node $v \in G$ into one of four types:

- **unvisited**: v has not yet been visited by the DFS.
- **fullyVisited**: v has been visited, and all its neighbors have also been visited.

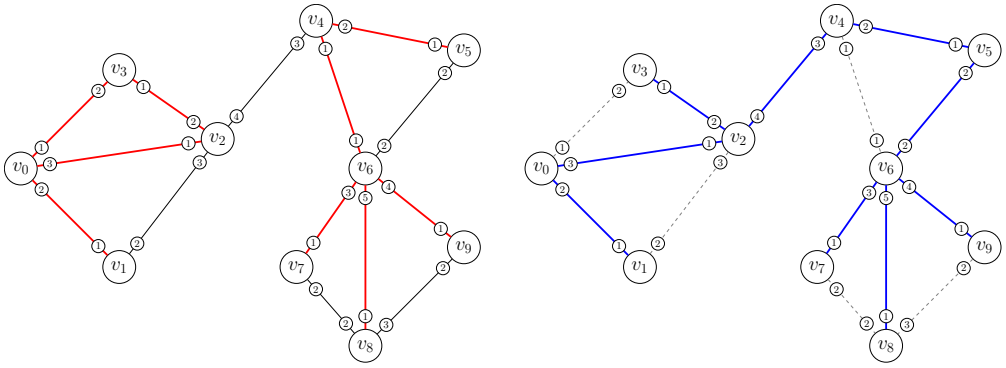


Figure 1 An example of a Port-One Tree. Left: Edges incident to a port 1 are highlighted in red. Right: A valid P1Tree is shown with tree edges in blue (solid) and non-tree edges in gray (dashed).

- **partiallyVisited**: v was reached via a `tpq` edge, and all of its unvisited neighbors are also reachable only via `tpq` edges.
- **visited**: v has been visited but is not **partiallyVisited** or **fullyVisited**.

`DFS_P1Tree()` prioritizes edges at any node in the order: `tp1` $\succ$ `t11` $\sim$ `t1q` $\succ$ `tpq`. The traversal proceeds like a standard DFS, moving forward to **unvisited** neighbors of **visited** nodes and backtracking from **fullyVisited** nodes. The key modification handles **partiallyVisited** nodes to ensure the P1Tree property. Suppose the DFS head reaches node u. The type of u is determined by inspecting its neighbors $N(u)$. Based on the type of u, the DFS head proceeds as follows:

(D0) All nodes are initially **unvisited**.

(D1) If u is **fullyVisited**, the DFS head backtracks to the parent of u.

(D2) If u is **visited**, the DFS head visits an **unvisited** neighbor of u along the highest priority edge.

(D3) If u is **partiallyVisited**, the DFS head backtracks to the parent of u.

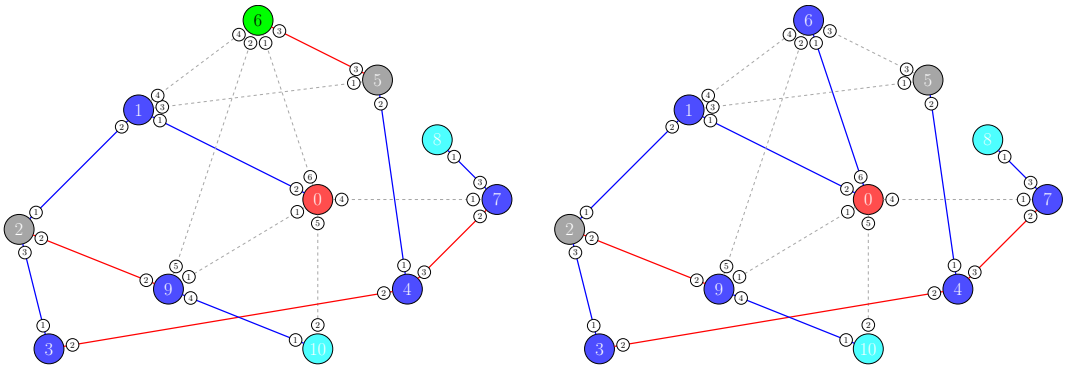
(D4) A **partiallyVisited** node u is treated as "re-visitable". When the DFS traversal later reaches u from a neighbor w via an edge of type `tp1` or `t11`, a *reconfiguration* occurs: the original parent edge of u (which was of type `tpq`) is removed from the tree, and the new edge $\{w, u\}$ is added. The parent of u is now swapped to w, and u's type is changed to **visited**.

This parent-swap operation is crucial, and swapping the parent of a leaf does not create a cycle. This mechanism guarantees that every node eventually acquires a port-1 incident edge in the final tree. Figure 2 illustrates this process.

▶ **Theorem 3.** `DFS_P1Tree()` *produces a* P1Tree $\mathcal{T}$ *of a port-labeled graph* G.

4 Agent-based P1Tree Construction

We now describe how mobile agents execute the `DFS_P1Tree()` algorithm to construct a P1Tree and disperse. The key challenge is performing neighborhood searches efficiently in an asynchronous environment to find the next edge to traverse. Our solution relies on selecting some nodes to be VACATED and using their settled agents as scouts for a fast parallel probing mechanism.



(a) DFS backtracks at ⑥ marking it **partiallyVisited**. Its parent edge $\{⑥, ⑤\}$ is `tpq` and its only **unvisited** neighbor ⑨ is connected via a `tpq` edge.

(b) Reconfiguration: the `tpq` edge $\{⑥, ⑤\}$ is removed and the edge $\{⓪, ⑥\}$ of type `tp1` is added, making ⓪ the new parent of ⑥.

Figure 2 Illustration of reconfiguration on a P1Tree $\mathcal{T}$. A `tpq` edge is swapped for an edge of type `tp1` or `t11`.

4.1 Executing the DFS with Agents

Agents collectively act as the "DFS head". Initially, all agents are at a root node v_0 and are *unsettled*. The agent with the highest ID settles at v_0. The remaining unsettled agents form a scout pool. To decide the next move, these scouts perform a neighborhood search. When the DFS head moves to an unvisited node, the highest-ID unsettled agent settles there. This process continues, with agents settling at new nodes, until all unsettled agents have settled. The reconfiguration of **partiallyVisited** nodes is handled by the settled agent at that node, which updates its parent information when visited by the DFS head from its port-1 neighbor.

4.2 Selecting Vacant Nodes for Scouting

To perform neighborhood searches in $O(1)$ epochs, we need multiple agents to probe neighbor ports in parallel. We achieve this by designating certain nodes as VACATED. The agent that would have stayed at a VACATED node instead joins the scout pool and travels with the DFS head. A node's state is either EMPTY (not yet visited), OCCUPIED (an agent is settled and remains there), or VACATED (settled agent is travelling with the DFS head). The decision to vacate is made locally based on the state of itself and its port-1 neighbor. Starting with an OCCUPIED root node, a node can become VACATED if its port-1 neighbor is occupied and it is not acting as the port-1 neighbor for a previously vacated node. These rules are designed to ensure that information about a vacated node can be retrieved from a nearby occupied node, which is essential for the probing mechanism described next.

▶ **Lemma 4.** *Let $\mathcal{T}$ be the partial tree of size k constructed by the agents. At least $\lfloor k/3 \rfloor$ of these k vertices are in state* VACATED.

4.3 Parallel Probing for $O(1)$-epoch Neighborhood Search

As established, the central challenge under asynchrony is distinguishing an empty node from a vacated one in constant time. Our `Parallel_Probe()` mechanism solves this directly. With a pool of scouts available (from vacated nodes and remaining unsettled agents), the DFS head can perform a neighborhood search in $O(1)$ epochs. `Parallel_Probe()` assigns available scouts to probe unexplored ports of the current node x. A scout agent a traversing

port p_{xy} to a neighbor y must determine if y is OCCUPIED, VACATED, or genuinely EMPTY (i.e., unvisited by any DFS). The scout agent determines a node x is empty by visiting the port-1 neighbor y of x, and the port-1 neighbor z of y if y is also empty. Each scout travels at most 3 edges away from x and returns, taking a constant number of moves. Since there are at least $\lceil (k-2)/3 \rceil$ scouts, all neighbors of a node can be probed in a constant number of parallel waves, thus taking $O(1)$ epochs.

▶ **Lemma 5.** `Parallel_Probe()` *at a node $x \in V$ correctly determines the state of all its neighbors in $O(1)$ epochs.*

5 The Optimal Asynchronous Dispersion Algorithm

The complete asynchronous dispersion algorithm, `RootedAsync()`, integrates the concepts of P1TREE construction, node vacating, and parallel probing. We first describe the rooted case and then extend it to general initial configurations.

5.1 Rooted Dispersion

The algorithm unfolds in two phases: exploration and retrace. In the exploration phase, agents build a P1TREE rooted at v_0 by traversing the graph and settling at nodes. At each node, the exploring DFS head checks if that node can be vacated or not. It settles an unsettled agent from the scout pool if the parallel neighborhood search finds an empty node. In the retrace phase, scouts return to their original nodes. The retrace works by having the agents move in a post-order traversal of the constructed P1TREE, detaching from the group when they reach their home nodes. The post-order traversal is possible due to the presence of parent information, the most recently visited child, and the immediate sibling of each settled agent. Given a group of settled agents, we prioritize the immediate sibling, then the most recently visited child. Afterwards, the child is detached. When no child remains, the agent group takes the port to reach the parent. Both phases take $O(k)$ epochs.

Summarizing the above discussion, we obtain the following results.

▶ **Theorem 6.** *The* `RootedAsync()` *algorithm achieves dispersion in $O(k)$ epochs with $O(\log k + \Delta)$ bits of memory per agent.*

5.2 General Dispersion

When agents start at multiple nodes, each group of co-located agents (a "multiplicity") initiates its own independent instance of the rooted dispersion algorithm. Each exploration is identified by a unique "treelabel" based on its root agent's ID. When two explorations meet (i.e., a scout from one tree probes a node occupied by an agent from another tree), a merger protocol is invoked. We call this `GeneralAsync()`. The merger strategy is based on a size-based subsumption rule. When exploration T_1 with k_1 agents meets T_2 with k_2 agents, the smaller exploration is absorbed by the larger one. For instance, if $k_1 < k_2$, all agents from T_1 (both settled and scouts) will abandon their exploration, gather at their root if possible, and then join the DFS head of T_2 as new unsettled agents. The larger exploration T_2 pauses briefly to absorb the new agents and then continues its DFS to find homes for them. This process ensures that the total number of agents in a single exploration grows monotonically. The overhead for collecting agents from an absorbed tree of size k_i is $O(k_i)$. A careful analysis shows that the total time spent in mergers across the entire execution is bounded by $O(k)$, preserving the overall optimal time complexity. From the above discussion, we have the following results.

▶ **Theorem 7.** *The* `GeneralAsync()` *algorithm achieves the dispersion in anonymous graph for arbitrary configuration in* $O(k)$ *epochs with* $O(\log k + \Delta)$ *bits of memory per agent.*

References

1 Ankush Agarwalla, John Augustine, William K Moses Jr, Sankar K Madhav, and Arvind Krishna Sridhar. Deterministic dispersion of mobile robots in dynamic rings. In *ICDCN*, pages 1–4, 2018. doi:10.1145/3154273.3154294.

2 John Augustine and William K. Moses Jr. Dispersion of mobile robots: A study of memory-time trade-offs. In *ICDCN*, pages 1:1–1:10, 2018. doi:10.1145/3154273.3154293.

3 Rik Banerjee, Manish Kumar, and Anisur Rahaman Molla. Optimizing robot dispersion on unoriented grids: With and without fault tolerance. In *ALGOWIN*, pages 31–45, 2024. doi:10.1007/978-3-031-74580-5_3.

4 Rik Banerjee, Manish Kumar, and Anisur Rahaman Molla. Optimal fault-tolerant dispersion on oriented grids. In *ICDCN*, pages 254–258, 2025. doi:10.1145/3700838.3700842.

5 Prabhat Kumar Chand, Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Fault-tolerant dispersion of mobile robots. In *CALDAM*, pages 28–40, 2023. doi:10.1007/978-3-031-25211-2_3.

6 Giuseppe F. Italiano, Debasish Pattanayak, and Gokarna Sharma. Dispersion of mobile robots on directed anonymous graphs. In *SIROCCO*, pages 191–211, 2022. doi:10.1007/978-3-031-09993-9_11.

7 Ajay D. Kshemkalyani. Dispersion of mobile robots on graphs in the asynchronous model. *Theor. Comput. Sci.*, 1044:115272, 2025. doi:10.1016/j.tcs.2025.115272.

8 Ajay D. Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *ICDCN*, pages 218–227, 2019. doi:10.1145/3288599.3288610.

9 Ajay D Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, Debasish Pattanayak, and Gokarna Sharma. Dispersion is (almost) optimal under (a) synchrony. In *SPAA*, 2025. doi:10.1145/3694906.3743317.

10 Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Fast dispersion of mobile robots on arbitrary graphs. In *ALGOSENSORS*, pages 23–40, 2019. doi:10.1007/978-3-030-34405-4_2.

11 Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots on grids. In *WALCOM*, pages 183–197, 2020. doi:10.1007/978-3-030-39881-1_16.

12 Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots using global communication. *J. Parallel Distributed Comput.*, 161:100–117, 2022. doi:10.1016/j.jpdc.2021.11.007.

13 Ajay D. Kshemkalyani and Gokarna Sharma. Near-optimal dispersion on arbitrary anonymous graphs. *Journal of Computer and System Sciences*, 152:103656, 2025. doi:10.1016/j.jcss.2025.103656.

14 Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Byzantine dispersion on graphs. In *IPDPS*, pages 942–951, 2021. doi:10.1109/IPDPS49936.2021.00103.

15 Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr. Optimal dispersion on an anonymous ring in the presence of weak byzantine robots. *Theor. Comput. Sci.*, 887:111–121, 2021. doi:10.1016/j.tcs.2021.07.008.

16 Debasish Pattanayak, Ajay D. Kshemkalyani, Manish Kumar, Anisur Rahaman Molla, and Gokarna Sharma. Optimal dispersion under asynchrony. *CoRR*, abs/2507.01298, 2025. doi:10.48550/arXiv.2507.01298.

17 Debasish Pattanayak, Gokarna Sharma, and Partha Sarathi Mandal. Dispersion of mobile robots in spite of faults. In *SSS*, pages 414–429, 2023. doi:10.1007/978-3-031-44274-2_31.

18 Yuichi Sudo, Masahiro Shibata, Junya Nakamura, Yonghwan Kim, and Toshimitsu Masuzawa. Near-linear time dispersion of mobile agents. In *DISC*, pages 38:1–38:22, 2024. doi:10.4230/LIPIcs.DISC.2024.38.

Brief Announcement: Proximal Byzantine Agreement: Improved Accuracy for Fault-Tolerant Replicated Datastreams

Roy Shadmon
University of California, Santa Cruz, CA, USA

Owen Arden
University of California, Santa Cruz, CA, USA

Abstract

Approximate Byzantine Agreement (ABA) protocols enable nonfaulty replicas with different initial values to derive a values within a ϵ-neighborhood of each other, despite the presence of Byzantine behavior. While they give strong guarantees for this ϵ-*agreement* property, they tend to have weaker guarantees that the derived value is *accurate* with respect to some ground truth. Worse, they often have impractical requirements such as large replica sets proportional to data dimensionality, or a priori knowledge of the maximum distance between nonfaulty values.

In *Stochastic Byzantine Agreement (SBA)*, the distribution of the nonfaulty values is the result of a stochastic process influenced by sensor measurement error or other sources of noise that affect system outputs. For these scenarios, we present *Proximal Byzantine Agreement* (PBA), a stochastic Byzantine agreement protocol which infers the most likely output of replicated computation based on the proposed values observed by each replica. Unlike ABA protocols, PBA prioritizes accuracy over agreement. PBA accuracy is relative to the variance of nonfaulty values, yielding comparatively more accurate results for noisy data, particularly when noise is asymmetric. Our evaluations demonstrate this accuracy scales with data dimensionality, outperforming or only mildly underperforming methods that require quorums with up to 10× *more replicas* and 4× *to* 124× *more computation time* per agreement decision, even at relatively low dimensions ($d = 4$ to $d = 18$).

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks; Computer systems organization → Sensor networks

Keywords and phrases Byzantine fault tolerance, distributed control systems, robust statistics

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.64

Funding Partial funding provided by 2025 Berkeley RDI AI & Decentralization Innovation Award.

1 Introduction

Reliably controlling systems and devices that interact with the physical environment is challenging in any context, but especially in edge networks. Volatile communication, power, and operating conditions demand greater fault tolerance, and edge devices and sensors are often more vulnerable to malicious manipulation than those in secure, monitored facilities. These realities complicate the coherent design of sensor data-processing pipelines, distributed control systems, and multi-agent robotics applications – scenarios that are *functionally* similar but *operationally* require ad hoc mitigation techniques to tolerate deployment conditions.

The growing demand for such systems spans consumer-facing devices such as home-automation sensors and controls, personal health monitoring devices and drug-delivery systems, and automotive telemetry and safety systems, as well as industrial and agricultural platforms for monitoring and controlling mechanical, physical, and biological processes. In all of these domains, the quality and reliability of sensor and sensor-derived data is central


© Roy Shadmon and Owen Arden;
licensed under Creative Commons License CC-BY 4.0

to effective operation. When that data is delayed, missing, or corrupted – whether from environmental noise, hardware faults, or malicious interference – the consequences can range from suboptimal performance to catastrophic, even unsafe, failures.

Despite shared characteristics, a divergence occurred between communities that prioritize agreement "regardless of what they agree upon" originally proposed by Lamport [9], and communities that also want guarantees on the accuracy of some quantity being approximated. Whereas most agreement protocols give *validity* [5] specifications on their output in terms of the initial values proposed by nonfaulty replicas, *accuracy* [12] specifications are with respect to a true quantity approximated by the outputs. Much subsequent work on approximate agreement does not explicitly consider accuracy, with some notable exceptions [1, 3] in the sensor fusion community. In all cases, relatively strong assumptions are made about nonfaulty values such as the maximum pairwise distance [10], maximum width of a containing interval [12], or the true value is contained in the intersection of ranges of nonfaulty replicas [3]. Violating these assumptions affects accuracy and can lead to arbitrary protocol failure.

This paper restores focus on the accuracy of an important subset of approximate Byzantine agreement (ABA) problems under weaker, more realistic assumptions. Since phenomena measured by sensors can often be modeled as stochastic processes [17], we define *Stochastic Byzantine Agreement (SBA)*, a refinement of the ABA problem. In the SBA problem, "true" values are generated by a stochastic process with an unknown *output distribution*, and the observations of those values proposed by nonfaulty replicas are perturbed by noise drawn from an unknown *error distribution*. Byzantine replicas act arbitrarily, and thus their proposed values are not generated – or even effectively approximated – by any distribution. Our SBA protocol, Proximal Byzantine Agreement (PBA), uses a novel *value selection process*, how replicas select outputs based on the messages they receive, that borrows techniques from robust statistics [7]. As long as a majority of these messages are stochastically distributed, the influence of arbitrarily selected values on the inference process is limited to a fixed bound.

This brief announcement reports on progress following our initial evaluation [18] of the feasibility of this value selection technique. Previously, only one-dimensional, normally distributed distributions were explored, and probabilistic protocol properties were described informally. Here we give formal, geometric definitions and theorems with rigorous guarantees, including evaluations comparing PBA to multiple ABA protocols on multidimensional data. These evaluations indicate a surprising degree of scalability with respect to dimensionality: only minor losses in accuracy compared to ABA protocols that require $10\times$ *more replicas* and $4\times$ *to* $124\times$ *more computation time* per agreement decision, even at relatively low dimensions ($d = 4$ to $d = 18$). Since each additional output offers more information from the unknown output distribution, our results indicate PBA can extract more information from observations, reinforcing the tradeoff [12] between high-precision agreement and statistical accuracy.

2 Proximal agreement as a probability maximization

Replicas participating in Approximate Byzantine Agreement (ABA) protocols (e.g.,[5, 14, 19]) start with an arbitrary real value and reach an output solution satisfying two properties:

1. ϵ-*Agreement*: all nonfaulty replicas eventually output values within ϵ of each other.
2. *Validity*: these values are bound (in some sense) by the initial values of nonfaulty replicas.

Mahaney and Schneider [12] distinguish Inexact Agreement (IA) from classical ABA by redefining the validity property in terms of *accuracy*. Outputs are instead bound by their distance to a true value $\hat{v}$, which each nonfaulty replica's initial value v_i approximates.

Our work concerns a new class of protocols that make an additional assumption, often satisfied by the underlying data processed by IA systems. Stochastic Byzantine Agreement (SBA) protocols require that $\hat{v}$ and nonfaulty v_i are – or can be effectively approximated as being – distributed according to probability distributions. No assumptions are made about the nature of Byzantine values. The two distributions are distinct and potentially unknown, but correlated since v_is approximate $\hat{v}$. SBA protocols also satisfy ϵ-agreement, but ϵ in SBA is larger than the ϵ values in classical ABA protocols. This difference in ϵ-agreement is by design and unavoidable due to tradeoffs between accuracy and high-precision agreement [12]. The goals of these protocols are close enough that we argue ABA is an appropriate umbrella term, with IA and SBA refining properties 1 and 2 as described above.

The key characteristic of our SBA protocol, *Proximal Byzantine Agreement* (PBA), is that each replica uses the set R of received values $v_i \in R$ to find a quorum of values r and a candidate $\hat{v}$ such that $\hat{v}$ would be the most likely true value if we knew that the quorum values were nonfaulty. Selecting the highest-probability $\hat{v}$ conditioned on the observations in r, ensures that if any (minority of) values in r turn out to be faulty, then the quorum r (and thus the inferred $\hat{v}$) is at least a likely as a quorum containing only nonfaulty values. At a high level, this process is described by the probability maximization in Def. 2.1.

▶ **Definition 2.1** (Proximal Byzantine Value Selection). *For received values $R \subseteq \mathbb{R}^n$ and quorum size s, select a set r from the s-sized subsets of R, and a value x that maximizes the likelihood that x is the true value given observations in r.*

$$\mathsf{PBA}(R, s) \triangleq \underset{\substack{r \in [R]^s \\ x \in \mathbb{R}^n}}{\operatorname{argmax}} P(x \mid r)$$

Our chosen approximation of Proximal Byzantine Value Selection (PBVS) uses the geometric median of each r as a robust estimator for the expected value of the unknown distribution of observations in r, thus approximating the candidate x. Below we will just use $\mathsf{PBA}(R, s)$ to refer to our approximation of PBVS to avoid additional notation. The geometric median [16] is well-suited for estimating the expected value of a broad class of distributions [15, 16], and remains robust even when $< \frac{1}{2}$ of the samples are arbitrarily corrupted. We compare the conditional likelihoods $P(x \mid r)$ for each r and x with a similarity-based approximation of conditional probabilities [2]. This is necessary for multidimensional data since the covariance matrix is unknown, but is a reasonable approach since each nonfaulty replica output is an approximation of the same underlying value.

More specifically, for each quorum r, we compute its median $x = \textsc{Gm}(r)$ and evaluate the likelihood $\widehat{P}(x \mid r)$ using a similarity measure adapted from Blok et al. [2]. $\widehat{P}(x \mid r)$ approximates the affect of the unknown covariance matrix and joint probability of q on the independent probability of x with an exponent α to score similarity between x and quorum values on a $[0, 1)$ scale. We chose this approach after discovering that computing the geometric median and then approximating its likelihood is significantly faster than iteratively evaluating $\widehat{P}(X = x \mid r)$ to find maximal values, and had almost no impact on accuracy.

2.1 Geometric properties

Approximation of a true value introduces a degree of uncertainty for systems that use those approximations to make decisions. We are unaware of any ABA protocols that estimates the uncertainty of its outputs.[1] In many scenarios, it may be better to perform no action at all than to act on very uncertain data. Therefore, calculating a concrete guarantee on each output is an important feature of PBA protocols.

[1] In a sense, the strong assumptions of existing IA protocols [12, 4, 3] act as *inputs* about uncertainty.

A well-known result [11] is that the displacement of the geometric median $\text{GM}(Q)$ of a set of nonfaulty values is bounded when up to $|F| < |Q|$ faulty values are added to the set.

▶ **Lemma 2.2** (Theoretical maximum displacement [6, 11]). *Let $R = Q \cup F$, $|Q| = s$ and $|F| = f$ such that $s > f$ with geometric medians $\text{GM}(Q) = m_Q$ and $\text{GM}(R) = m_R$. Let $\Delta^Q = \max_{q_i \in Q} ||q_i - \text{GM}(Q)||_2$ and $C_0^{(s,f)} = \frac{s}{\sqrt{s^2 - f^2}}$. Then the maximum displacement is*

$$||m_Q - m_R||_2 \leq C_0^{(s,f)} \Delta^Q$$

However, the contents of Q is unknown, so the bound implied by Lemma 2.2 cannot be directly calculated from the information available. Instead, Lemma 2.3 presents a computable bound guaranteed to contain the geometric median of the nonfaulty values, but is based only on the received values and the system parameters (i.e., n, f, and s).

▶ **Lemma 2.3** (Computable maximum displacement). *Let $R = Q \cup F$, $|Q| = s$ and $|F| = f$ such that $s > 2f$ with $\text{GM}(Q) = m_Q$ and $\text{GM}(R) = m_R$, and $C^{(s,f)} = \frac{s}{\sqrt{2sf - f^2}}$. Then*

$$||m_Q - m_R||_2 \leq C^{(s,f)} \Delta_f^R$$

The computable displacement is used to compute the *region guarantee* for a given PBA result. The fact that PBA only produces outputs with region guarantees containing the true output is our instantiation of an *accuracy* [12] property.

▶ **Definition 2.4** (Region Guarantee). *For any set $R \subseteq \mathbb{R}^d$, $|R| > s + f$ where $s > f$, define $R^{(f,x)} \subseteq R$ by removing the f elements of R furthest from x. Let $\Delta_f^R = \max_{r_i \in R^{(f, \text{GM}(R))}} ||r_i - \text{GM}(R)||_2$. The* region guarantee *$RG(R, s, f)$ is a d-dimensional ball centered at x with radius $2C^{(s,f)}\Delta_f^R$.*

▶ **Lemma 2.5** (Accuracy). *For any $R = (Q \cup F)$ containing $|Q| = s > 2f$ nonfaulty and at most $|F| = f$ faulty values, let $\mathsf{PBA}(R, s) = (x, R')$. Then $\text{GM}(Q) \in RG(R, s, f)$ and*

$$||m_Q - x||_2 \leq C_0^{(s,f)} \Delta^{R'} + C_0^{(s,f)} \Delta^Q \leq 2C^{(s,f)} \Delta_f^R$$

2.2 OneShot PBA Agreement and Termination

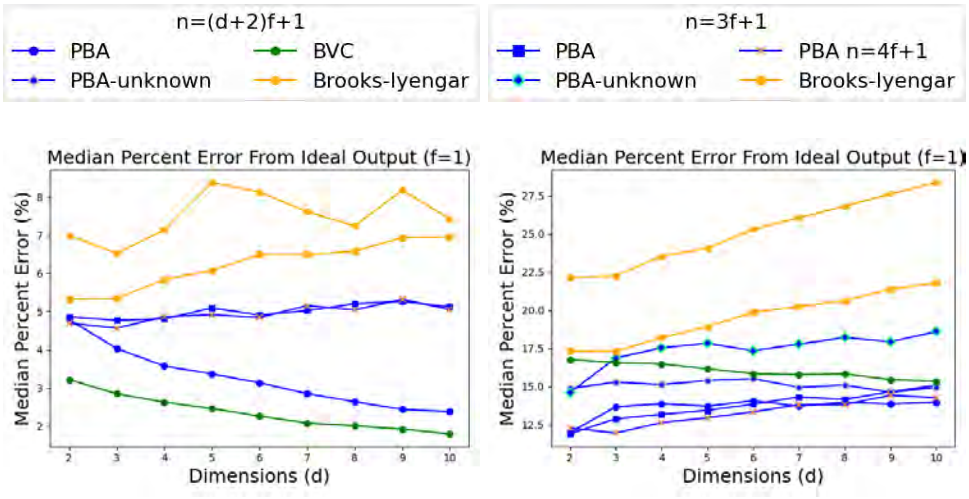
For this brief announcement, we describe OneShot PBA, a protocol that prioritizes accuracy. It infers the potential distances between the true value, its output, and other replica outputs using only the messages it receives, without additional coordination. Clients can receive proposed values directly from replicas and execute our PBVS algorithm locally to select values. Like most IA protocols [3, 4, 13], but unlike other ABA protocols [5, 12, 14, 19], OneShot does not iterate to converge on a single answer. Doing so would obscure the initial values, sacrifice accuracy, and undermine the fidelity of the region guarantees.

The termination conditions are thus straightforward: OneShot PBA requires $s + f$ values where $s \geq 2f + 1$, so termination for the asynchronous case requires at least $n \geq s + 2f \geq 4f + 1$ replicas. Otherwise, f faulty replicas could withhold values without being distinguished from f nonfaulty replicas experiencing a network partition, and halt progress.

PBA OneShot's requirements on replica set size compare favorably to other ABA protocols (Table 1), particularly when considering its lack of dependency on data dimensionality and weak assumptions regarding the range of correct values. Unlike prior work that tend to improve accuracy with more replicas, a distinguishing characteristic of PBA value selection is that more replicas are not required to ensure the safety or liveness properties of the system, yet still benefits from the increase in accuracy.

Table 1 Replica set size. d is number of data dimensions.

Algorithm	$n \geq$
OneShot	$4f + 1$
ABA [5]	$5f + 1$
IA [12]	$3f + 1$
BI [3, 4]	$2df + 1$
BVC [14, 19]	$(d + 2)f + 1$



(a) Gaussian samples, Byzantine attack. **(b)** Exponential samples, Byzantine attack.

Figure 1 Empirical simulations measuring PBA's accuracy compared to baseline algorithms given data distributed by a Gaussian (symmetric) and Exponential (asymmetric) distributions.

Despite the absence of inter-replica coordination or iteration, the PBA values derived by consumers from different subsets are bounded with respect to their distance to the true expected value. This bound is in terms of the displacement properties of the geometric median discussed in §2.1, and contingent on quorum sizes representing a sufficiently large fraction of the nonfaulty replicas, in this case $> \frac{|Q|}{2}$.

It is taken for granted in other protocols that quorums must contain at least half of all nonfaulty replicas since these protocols ensure all quorums intersect. We only make this size requirement explicit for agreement because not all instances of PBA necessarily involve multiple consumers. For example, there is no need for agreement if a system controller is the only consumer of replica outputs, as is often the case in industrial control systems. For contrast, note that accuracy (Lemma 2.5) only requires quorum membership to be more than twice the number of faulty replicas.

▶ **Lemma 2.6** (Agreement). *For any $R^i = (Q^i \cup F^i) \subseteq R = (Q \cup F)$, let $r = |R^i|$, $s = |Q^i|$, $f = |F|$, and $PBA(R^i, s) = (x_i, R^i_*)$. If $s > \left\lceil \frac{|R|}{2} \right\rceil$ and $r \geq s + f$, then for any other subset $R^j = (Q^j \cup F^j) \subseteq R$ where $|R^j| \geq r$ and $PBA(R^j, s) = (x_j, R^j_*)$, $||x_i - x_j|| \leq 2C_0^{(s,f)}\Delta^Q$.*

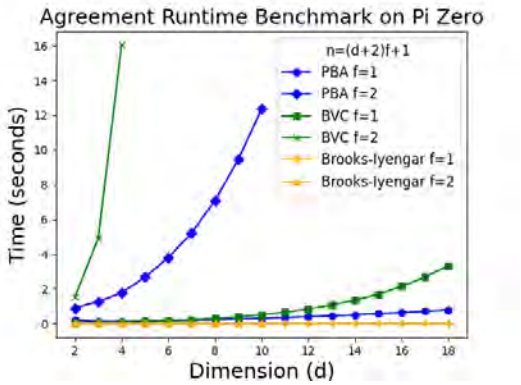


Figure 2 Runtime benchmark (Pi Zero).

3 Evaluation

PBA accuracy compared to baseline. We perform repeated simulations on Chameleon Cloud [8], comparing PBA under several quorum sizes against approximate Byzantine Vector Consensus (BVC) [19] and Brooks-Iyengar (BI) [4]. Here we present only the Byzantine attack scenario. Each trial samples $n - f$ d-dimensional replica outputs x_i from a random Gaussian or Exponential distribution (with a fixed, low variance) and multiplies each element-wise by a noise vector y_i independently sampled from an error distribution with $E[Y] = 1$ and (uniformly-sampled) standard deviation from 0.01 to 0.09. We compute the agreement vector of each protocol and report the median percent error with respect to x_i across 1000 trials.

Figure 1a compares results when $n - f$ outputs are sampled from a Gaussian distribution under an optimal Byzantine attack with $f = 1$. While BVC remains highly robust, PBA achieves comparable accuracy with just $n = 4f + 1 = 5$ (small n) outputs, versus BVC's $n = (d+2)f + 1 = 13$ (large n) outputs. This indicates that at least some of BVC's accuracy may result from its reliance on more samples, but any ABA protocol will improve in accuracy with more samples: aggregating values concentrates outputs around the true value. PBA approaches the true value with fewer samples (i.e., replicas) and performance overhead.

In the Exponential and large n setting ($n = (d+2)f+1$), Figure 1b shows PBA outperforms all other $n = 4$ protocols. In the small n setting, PBA demonstrates superior accuracy to all baselines – even in the PBA-unknown case, where the prior distribution is estimated as Gaussian. This accuracy advantage likely stems from PBA's quorum selection strategy and agreement inference method. In contrast, the benefit other protocols received from additional samples in the Gaussian case, a symmetric distribution, are tempered in the exponential case, an asymmetric distribution. The gap in accuracy suggests the averaging effect occurring in other protocols generalizes poorly to other distributions, whereas PBA's more statistically-informed approach adapts more easily.

Runtime performance. Figure 2 presents runtime benchmarks for $f = 1$ and $f = 2$ on a node hosted on a single Raspberry Pi Zero 2W. These results are based on repeated simulations of the PBA, BVC, and BI protocols operating on sets of n d-dimensional vectors. where $n = (d+2)f + 1$, and PBA quorum sizes set to $s = n - f$. PBA is faster than BVC for $f = 1$, $d > 5$ (with similar performance at lower d), and more than $4\times$ faster at $f = 1$, $d = 18$. In all $f = 2$, $d = 4$ scenarios, PBA significantly outperforms BVC ($124\times$). While BI also exhibits scalable runtime in high-dimensional settings, each replica must produce outputs

with a known error range that contains the true value, and that must also intersect the ranges of all other nonfaulty nodes. If these invariants are violated by any nonfaulty replica, the system may not withstand Byzantine attack. Even without an attack, the protocol could fail to terminate.

Our empirical results against BVC reinforces that PBA offers a compelling trade-off: it maintains competitive accuracy while substantially reducing computational cost. Beyond performance, PBA also provides robust fault tolerance and accurate agreement with significantly fewer replica outputs. Notably, PBA's minimum required number of replicas and quorum size are independent of d; in contrast to BVC, whose replica requirements scale significantly with d. Under-provisioned PBA systems instead have more uncertainty, but the amount is characterized by region guarantees. This scalable perfomance and graceful, principled degradation indicate promise for both large-scale and resource-constrained sensor-based control systems and datastream processing applications.

References

1 Buke Ao, Yongcai Wang, Lu Yu, Richard R. Brooks, and S. S. Iyengar. On Precision Bound of Distributed Fault-Tolerant Sensor Fusion Algorithms. *ACM Computing Surveys (CSUR)*, May 2016. doi:10.1145/2898984.

2 Sergey Blok, Douglas Medin, and Daniel Osherson. Probability from similarity. In *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*, pages 36–42, 2003.

3 Richard Ree Brooks and S. Sitharama Iyengar. Optimal matching algorithm for multidimensional sensor readings. In *Sensor Fusion and Networked Robotics VIII*, volume 2589. SPIE, September 1995. doi:10.1117/12.220948.

4 R.R. Brooks and S.S. Iyengar. Robust distributed computing and sensing algorithm. *Computer*, 29(6), June 1996. doi:10.1109/2.507632.

5 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of The Acm*, 33(3), May 1986. doi:10.1145/5925.5931.

6 El-Mahdi El-Mhamdi, Sadegh Farhadkhani, Rachid Guerraoui, and Lê-Nguyên Hoang. On the Strategyproofness of the Geometric Median. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. PMLR, April 2023. URL: https://proceedings.mlr.press/v206/el-mhamdi23a.html.

7 Peter J. Huber and Elvezio M. Ronchetti. *Robust Statistics*. Wiley, 2011.

8 Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020. URL: https://www.usenix.org/conference/atc20/presentation/keahey.

9 Leslie Lamport. The weak Byzantine generals problem. *Journal of the ACM*, 30(3), 1983. doi:10.1145/2402.322398.

10 Leslie Lamport and P. M. Melliar-Smith. Byzantine clock synchronization. In *Principles of Distributed Computing (PODC)*, 1984. doi:10.1145/800222.806737.

11 Hendrik P. Lopuhaa and Peter J. Rousseeuw. Breakdown Points of Affine Equivariant Estimators of Multivariate Location and Covariance Matrices. *The Annals of Statistics*, 19(1), March 1991. doi:10.1214/aos/1176347978.

12 Stephen R. Mahaney and Fred B. Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *Principles of Distributed Computing (PODC)*. ACM Press, 1985. doi:10.1145/323596.323618.

13 Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4), November 1990. doi:10.1145/128733.128735.

14 Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. ACM, June 2013. doi:10.1145/2488608.2488657.

15 Stanislav Minsker. Geometric median and robust estimation in Banach spaces. *Bernoulli*, 21(4), November 2015. doi:10.3150/14-BEJ645.

16 Stanislav Minsker and Nate Strawn. The Geometric Median and Applications to Robust Mean Estimation. *SIAM Journal on Mathematics of Data Science*, 6(2), June 2024. doi:10.1137/23M1592420.

17 Emanuel Parzen. *Stochastic Processes*. Dover Books on Mathematics. Dover Publications Inc., 2015.

18 Roy Shadmon and Owen Arden. Enhancing accuracy in approximate byzantine agreement with bayesian inference. In *55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2025. doi:10.1109/DSN-S65789.2025.00064.

19 Nitin H Vaidya and Vijay K Garg. Byzantine vector consensus in complete graphs. In *Proc. of the 2013 ACM Symposium on Principles of Distributed Computing*, 2013.

Brief Announcement: Congested Clique Counting for Local Gibbs Distributions

Joshua Z. Sobel
Department of Computer Science, University of Iowa, Iowa City, IA, USA

Abstract

There are well established reductions between combinatorial sampling and counting problems (Jerrum, Valiant, Vazirani TCS 1986). Building off of a very recent parallel algorithm utilizing this connection (Liu, Yin, Zhang arxiv 2024), we demonstrate the first approximate counting algorithm in the CongestedClique for a wide range of problems. Most interestingly, we present an algorithm for approximating the number of q-colorings of a graph within ϵ-multiplicative error, when $q > \alpha\Delta$ for any constant $\alpha > 2$, in $\tilde{O}\left(\frac{n^{1/3}}{\epsilon^2}\right)$ rounds. More generally, we achieve a runtime of $\tilde{O}\left(\frac{n^{1/3}}{\epsilon^2}\right)$ rounds for approximating the partition function of Gibbs distributions defined over graphs when simple locality and fast mixing conditions hold. Gibbs distributions are widely used in fields such as machine learning and statistical physics. We obtain our result by providing an algorithm to draw n random samples from a distributed Markov chain in parallel, using similar ideas to triangle counting (Dolev, Lenzen, Peled DISC 2012) and semiring matrix multiplication (Censor-Hillel, Kaski, Korhonen, Lenzen, Paz, Suomela PODC 2015). Aside from counting problems, this result may be interesting for other applications requiring a large number of samples.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Random walks and Markov chains

Keywords and phrases Distributed Sampling, Approximate Counting, Markov Chains, Gibbs Distributions

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.65

Related Version *Full Version*: https://arxiv.org/abs/2508.13083

Funding The author was partially supported during this project by National Science Foundation grants 1955939 and 2402835.

Acknowledgements Thanks to Sriram Pemmaraju for suggestions and discussions.

1 Introduction

In a pioneering work, Valiant proposed the computational complexity class #P [12]. Problems in #P are the counting variants of decision problems in NP; for example, counting the number of proper vertex colorings of a graph using at most q colors (q-colorings). Unfortunately, Valiant established that even when a decision problem is in P, such as determining whether a bipartite graph has a perfect matching, the corresponding counting problem can be NP-hard. Thus for most counting problems, we need to settle for ϵ multiplicative approximation rather than exact counting. The goal is to estimate a true count X by $\hat{X}$, such that $(1-\epsilon)X \leq \hat{X} \leq (1+\epsilon)X$ with probability at least $\frac{3}{4}$. By the standard median trick, 3/4 can be boosted to any constant less than one.

The typical method for approximate counting involves drawing random samples that are solutions to the corresponding search problem. In other words, we can count the number of colorings of a graph by sampling random colorings. This approach dates back to Jerrum, Valiant, and Vazirani. The initial reduction from counting to sampling [8] requires drawing a very large number of samples; this has been improved over several subsequent works [7, 1, 13, 10]. In particular, when considering Gibbs distributions defined over graphs with n


© Joshua Z. Sobel;
licensed under Creative Commons License CC-BY 4.0



vertices and meeting mild conditions, Štefankovič, Vempala, and Vigoda [13] showed that counting can be accomplished by taking $\tilde{O}\left(\frac{n}{\epsilon^2}\right)$ samples. Gibbs distributions originated in physics, and generalize many problems such as the uniform distribution of q-colorings.

Unfortunately, the algorithm from [13] cannot be fully parallelized, as samples drawn later depend on samples drawn earlier. Very recently, Liu, Yin, and Zhang [10] also showed that for many Gibbs distributions defined over graphs counting can be done using $\tilde{O}\left(\frac{n}{\epsilon^2}\right)$ total samples, but in their approach every sample can be taken in parallel. This implies that counting in the CongestedClique reduces to quickly drawing $\tilde{O}\left(\frac{n}{\epsilon^2}\right)$ samples. From existing work, [5, 6, 4], we know that for many Gibbs distributions drawing a single sample within total variation distance error of δ can be done in $O(\log \frac{n}{\delta})$ CongestedClique rounds, using the *distributed Metropolis-Hastings* Markov chain. For Gibbs distributions that we call *local*, we identify a novel structural similarity between simulating one transition of n instances of the Markov chain and semiring matrix multiplication. In particular, utilizing the approach of Dolev, Lenzen, and Peled [3] for triangle counting and Censor-Hillel, Kaski, Korhonen, Lenzen, Paz, and Suomela [2] for semiring matrix multiplication, we can simulate one transition of n instances of the Markov chain in $O(n^{1/3})$ rounds and thus often obtain an approximate count in $\tilde{O}\left(\frac{n^{1/3}}{\epsilon^2}\right)$ CongestedClique rounds. The full version of our paper contains all of our results and proofs.

1.1 Main Results

Our central contribution is an algorithm to draw many samples at once from the distributed Metropolis-Hastings Markov chain. Beyond counting, this result may be useful in other algorithms where several samples are needed.

▶ **Theorem 1.** *One transition can be simulated for n instances of the distributed Metropolis-Hastings chain in $O(n^{1/3})$ rounds.*

As a corollary to the previous theorem and the results of [10], under mild locality and mixing conditions which we will describe later, there is an $\tilde{O}\left(\frac{n^{1/3}}{\epsilon^2}\right)$ round counting algorithm. As the most interesting example, we obtain a fast algorithm for approximating the number of q-colorings of an input graph with degree Δ. In addition, our full paper demonstrates a much faster algorithm for the hardcore model with low fugacity.

▶ **Corollary 2.** *There is an algorithm for approximating the number of q-colorings of a graph in $\tilde{O}(\frac{n^{1/3}}{\epsilon^2})$ rounds, when $q > \alpha\Delta$, for $\alpha > 2$, within multiplicative error ϵ, with probability at least $\frac{3}{4}$.*

We also give a conditional lower bound for approximating a partition function, Theorem 10. Even under our locality and mixing assumptions, counting when $\epsilon \leq \frac{1}{32n^2}$ is as hard as triange detection.

2 Technical Preliminaries

2.1 Gibbs Distributions on Graphical Models

Consider a traditional graph labeling problem, where each vertex receives a label from some underlying alphabet $\mathcal{A}$. We focus on randomly sampling from a subset, $\mathcal{S}$, of such labelings, given a distribution. In particular, we want to sample from a Gibbs distribution. Borrowing terminology from physics, we have the Hamiltonian, a function $H : \mathcal{S} \to \{0, 1, ..., h\}$, for some integer $h \geq 0$.

▶ **Definition 3** (Gibbs Distribution). *The Gibbs distribution at inverse temperature $\beta \geq 0$ assigns a labeling, $\sigma \in \mathcal{S}$, a probability proportional to $\exp(-\beta H(\sigma))$.*

In the trivial case, if we choose $H = 0$, we get a uniform distribution over $\mathcal{S}$.

▶ **Definition 4** (Partition Function). *The normalizing constant of the distribution, $Z(\beta) = \sum_{\sigma \in \mathcal{S}} \exp(-\beta H(\sigma))$, is referred to as the partition function.*

We will make the assumption that $|\mathcal{A}|$ and h are bounded by a polynomial in n. This is not a significant restriction and is necessary both algorithmically and to ensure messages are of size $O(\log n)$.

For this paper we will focus heavily on the well studied Potts model.

▶ **Definition 5.** *The Potts model is a Gibbs distribution where $\mathcal{A} = [q]$, is a set of colors. $\mathcal{S}$ is $[q]^V$. The Hamiltonian, H, assigns to each, possibly improper, coloring the number of monochromatic edges (edges where both vertices have the same color).*

Note that when $\beta = \infty$, the Potts model simply becomes the uniform distribution over proper colorings.[1] Furthermore, $Z(\infty)$ counts the number of proper colorings. On the other hand, when $\beta = 0$, the model is uniform over all proper and improper colorings. Then $Z(0) = q^n$. More precisely, when $\beta > 0$, this model is called the antiferromagnetic Potts model, since neighboring vertices favor not having the same color.

2.2 Counting From Sampling

We would now like to estimate $Z(\beta)$, for a given choice of β. We call this the generalized counting problem, as Z gives the weighted count over all labelings. As discussed in the introduction, it is often sufficient to take $\tilde{O}(\frac{n}{\epsilon^2})$ samples. More precisely, the result from Liu, Yin, and Zhang [10] is summarized by the next theorem.

▶ **Theorem 6.** *For a Gibbs distribution and inverse temperatures $0 \leq \beta_1 < \beta_2$, $\frac{Z(\beta_2)}{Z(\beta_1)}$ can be computed with a multiplicative error of ϵ with probability at least $\frac{3}{4}$ given the Hamiltonian of $\tilde{O}(\frac{n}{\epsilon^2})$ samples taken from the Gibbs distribution for inverse temperatures ranging from $[\beta_1, \beta_2]$. Furthermore, the temperature each sample is taken at can be precomputed and only depends on n, h, and $|\mathcal{A}|$.*

The theorem implies that we can approximate $Z(\beta)$, as long as $Z(\beta')$ is known in advance for some $\beta' \geq 0$ and that we can draw $\tilde{O}(\frac{n}{\epsilon^2})$ samples for inverse temperatures in the range $[\min(\beta, \beta'), \max(\beta, \beta')]$.

2.3 Pairwise Markov Random Fields

The traditional way to sample from a Gibbs distribution is using a rapidly mixing Markov chain. We begin from an arbitrary state (a graph labeling) and simulate transitions until the chain is *mixed*, meaning that the current state is roughly drawn from the desired distribution. Here we will focus on sampling from the probability space defined by a pairwise Markov random field. A pairwise Markov random field on a graph consists of vertex constraints and edge constraints.

[1] We treat the infinite case in the sense of the limit as $\beta \to \infty$.

For each vertex v, we have a constraint $b_v : \mathcal{A} \to \mathbb{R}_{\geq 0}$. For each edge $e = (u, v)$ we also have a constraint $A_e : \mathcal{A} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$. Here the edges are undirected, but the function A_e is not necessarily symmetric. The probability of graph vertex labeling l is defined to be proportional to

$$\prod_v b_v(l(v)) \cdot \prod_{e=(u,v)} A_e(l(u), l(v))$$

▶ **Definition 7** (Local Gibbs Distribution). *We call a Gibbs distribution local if*

1. *The distribution can be represented as a pairwise Markov random field.*
2. *The Hamiltonian H can be represented as a sum over sub-functions defined on each edge and vertex. $H(\sigma) = \sum_v H_v(\sigma) + \sum_{e=(u,v)} H_e(u, v)$. Furthermore, each of these subfunctions also has a codomain of $[h]$.*
3. *An element $\sigma \in \mathcal{S}$ can be constructed in the* CongestedClique *in $O(\log n)$ rounds.*

The third condition is needed to initialize the Markov chain.

▶ **Example 8** (Potts Model). We can see that the Potts model is a local Gibbs distribution. For convenience we make the change of variable, $\beta = -\ln \lambda$. We choose $b_v(x) = 1$ and $A_e(x, y) = \begin{cases} \lambda, \text{ if } x = y \\ 1, \text{else} \end{cases}$

2.4 Distributed Sampling

Feng, Sun, and Yin [5] found a preliminary distributed Markov chain to sample from pairwise Markov random fields. In the case of q-colorings, this was improved simultaneously by Fischer and Ghaffari [6] and Feng, Hayes, and Yin [4]. The improved chain can be generalized beyond colorings, see [11] by Pemmaraju and Sobel for the chain explicitly given in full generality. One transition of the chain, for some yet to be determined probability p, is defined in the following pseudocode. Every vertex v has its current label X_v. For ease of notation, in the edge acceptance step, any fraction that is not well defined (because certain vertices may not be active) equals 1. Note that on some pairwise Markov random fields this chain may have a very large mixing time, $\gg \log n$ and possibly infinite.

Algorithm 1 Distributed Metropolis-Hastings.

for each vertex v **do**
  v becomes active with probability p
  if v is active **then**
    v samples a proposed label, σ_v, from a distribution proportional to $b_v(\sigma)$
  end if
end for
for each edge $e = (u, v)$ **do**
  edge e is accepted with probability $\frac{A_e(X_u, \sigma_v)}{\max A_e} \cdot \frac{A_e(\sigma_u, X_v)}{\max A_e} \cdot \frac{A_e(\sigma_u, \sigma_v)}{\max A_e}$
end for
for each active vertex v **do**
  $X_v \leftarrow \sigma_v$, if every edge incident to v is accepted
end for

3 Our Results

3.1 Fast Mixing for the Potts Model

For the Potts model, the distributed Metropolis-Hastings Markov chain has a mixing time of $O(\log \frac{n}{\delta})$. As the proof technique is standard, we defer the proof to the appendix of the full version of our paper.

3.2 Fast Simulation of Distributed Metropolis-Hastings

We would now like to estimate the partition function of a local Gibbs distribution. As described earlier, we need to know the Hamiltonian of $k = \tilde{O}(\frac{n}{\epsilon^2})$ samples from the Gibbs distribution. This can be found by running k instances of the distributed Metropolis-Hastings Markov chain. We can simulate one transition of n instances of the chain in parallel.

▶ **Theorem 1.** *One transition can be simulated for n instances of the distributed Metropolis-Hastings chain in $O(n^{1/3})$ rounds.*

3.2.1 Algorithm

Initialization

Each chain starts at an initial labeling. At time step t, each vertex will hold $X_{v,i,t}$, the label of vertex v in chain i. It will also sample its new proposed label, $\sigma_{v,i,t}$. In the case the vertex is inactive in a chain, $\sigma_{v,i,t} = \sqcup$.

To decide if $X_{v,i,t+1} \leftarrow \sigma_{v,i,t}$, assuming $\sigma_{v,i,t} \neq \sqcup$, v needs to know if any of its incident edges are rejected in chain i. We will use the technique from Dolev, Lenzen, and Peled [3] for triangle counting and also used by Censor-Hillel, Kaski, Korhonen, Lenzen, Paz, Suomela [2] for matrix multiplication over semirings.

For convenience, we will use 0 based indexing. Let A be the adjacency matrix of the graph. Consider an $n \times n \times n$ cube, Q, where $Q_{v,w,i}$ is true iff $A_{v,w} = 1$ and $\{v, w\}$ is not accepted in chain i. Furthermore, think of Q evenly divided into n, $n^{2/3} \times n^{2/3} \times n^{2/3}$ subcubes. Let subcube $Q[x, y, z]$ refer to the subcube over the indices $[xn^{2/3}, (x+1)n^{2/3}) \times [yn^{2/3}, (y+1)n^{2/3}) \times [zn^{2/3}, (z+1)n^{2/3})$. We will use a similar notation for matrices, $A[x, y]$ refers to the submatrix over the indices $[xn^{2/3}, (x+1)n^{2/3}) \times [yn^{2/3}, (y+1)n^{2/3})$.

Computing Q

Each subcube is arbitrarily assigned to a machine that will be responsible for generating the entries of the subcube. Because edges are non-directional, we assign the subcubes $Q[x, y, z]$ and $Q[y, x, z]$ to the same machine. Each machine is assigned at most two subcubes.

Consider the machine M holding the subcubes $Q[x, y, z]$ and $Q[y, x, z]$. To fill in the subcubes, M needs to hold $A[x, y]$, $\sigma[x, z]$, $\sigma[y, z]$, $X[x, z]$, and $X[y, z]$. Altogether, M needs to receive $\Theta(n^{4/3})$ words. For every machine to do this step in parallel, each machine needs to send and receive $\Theta(n^{4/3})$ words. This communication is done in $\Theta(n^{1/3})$ rounds, using Lenzen's result [9]. M then uses the information it received in addition to its randomness to fill in the (possibly two) subcube(s) it is responsible for. More precisely, M runs the edge acceptance step of the Markov chain for the edge-chain pairs it is responsible for and fills in the subcube(s).

Updating Labels

M will compress each subcube it holds into a submatrix by applying union along the y-axis. In particular, the subcube $Q[x, y, z]$ is flattened into the submatrix $R^y[x, z]$, where $R^y[x, z]_{i,k} =$ true if there exists j such that $Q[x, y, z]_{i,j,k} =$ true.

Finally, vertex u accepts its proposal in chain i if $\vee_y R^y_{u,i} =$ false, as this indicates that no incident edge to u was rejected. To compute this for every chain, u receives $R^y_{u,i}$ for all y and i. Doing this simultaneously for all vertices, requires each machine to send and receive $O(n^{4/3})$ words of information. This takes a final $O(n^{1/3})$ rounds.

▶ **Corollary 9.** *With success probability at least $\frac{3}{4}$, we can approximate the partition function, $Z(\beta)$, of a local Gibbs distribution with multiplicative error ϵ in $\tilde{O}(\frac{n^{1/3}}{\epsilon^2})$ rounds, when $Z(\beta')$ is known for some β' and the associated distributed Metropolis-Hastings chain has an $O(\log n)$ mixing time for inverse temperatures in the interval $[\min(\beta, \beta'), \max(\beta, \beta')]$.*

Proof. Theorem 6 and Theorem 1 almost immediately give us the result. We draw $\tilde{O}(\frac{n}{\epsilon^2})$ samples from the Gibbs distribution by simulating $O(\log n)$ transitions of the Markov chain. There is one slight problem, the samples are held in a distributed fashion and we need the Hamiltonian of each sample held centrally at some machine. We use a very similar strategy as before, with a slight change described in our full paper. ◀

▶ **Corollary 2.** *There is an algorithm for approximating the number of q-colorings of a graph in $\tilde{O}(\frac{n^{1/3}}{\epsilon^2})$ rounds, when $q > \alpha\Delta$, for $\alpha > 2$, within multiplicative error ϵ, with probability at least $\frac{3}{4}$.*

3.3 Lower Bound

We can show that even in the case where the distributed Metropolis-Hastings chain has mixing time $O(\log \frac{n}{\delta})$, we still have conditional hardness results.

Consider the following Gibbs distribution. $\mathcal{A} = V \cup [3n]$. $\mathcal{S}$ are the labelings such that every vertex's label is either in $[3n]$ or is a neighboring vertex. We let the Hamiltonian count the number of edges $\{u, v\}$ such that the labels of u and v are the same vertex. It is not hard to show that this is a local Gibbs distribution and that the mixing time of the distributed Metropolis-Hastings chain is $O(\log \frac{n}{\delta})$ for $\beta \geq 0$.

As always, when $\beta = 0$, all labelings in $\mathcal{S}$ are equally likely. $Z(0) = \prod_v 3n + \deg(v)$. On the other hand, when $\beta = \infty$, we remove labelings where two neighboring vertices are labeled the same mutually neighboring vertex. This can only happen if there is a triangle in the graph. Thus, $Z(0) > Z(\infty)$ if and only if there is a triangle in the graph. In fact, if the graph has at least one triangle, a, b, c, then $1 - \frac{Z(\infty)}{Z(0)} \geq \frac{1}{16n^2}$, since that is a lower bound on the probability that vertices a and b both choose c in a uniform random labeling. Rearranging, $Z(\infty) \leq Z(0)(1 - \frac{1}{16n^2}) \leq Z(0)(1 - \frac{1}{32n^2})^2$.

Note that we can easily compute $Z(0)$ exactly. We can choose $\epsilon = \frac{1}{32n^2}$ and take an approximate count of $Z(\infty)$, $\bar{Z}$. Suppose the graph doesn't have a triangle. Then, $Z(\infty) = Z(0)$. Thus, $\bar{Z} \geq (1 - \frac{1}{32n^2})Z(0)$. Otherwise, the graph has a triangle. Then, $\bar{Z} \leq (1 + \frac{1}{32n^2})Z(\infty) \leq (1 - \frac{1}{32n^2})^2(1 + \frac{1}{32n^2})Z(0) < (1 - \frac{1}{32n^2})Z(0)$. Thus, we can use $\bar{Z}$ to determine if the graph has a triangle.

▶ **Theorem 10.** *Approximating the partition function of a local Gibbs distribution when $\epsilon \leq \frac{1}{32n^2}$ with success probability p is as hard as triangle detection with success probability p. This is true even when samples can be taken for all $\beta \geq 0$ in $\tilde{O}(1)$ rounds.*

Note that the best known algorithm for triangle detection takes $O(n^{0.157})$ rounds [2, 14].

References

1. Ivona Bezáková, Daniel Štefankovič, Vijay V. Vazirani, and Eric Vigoda. Accelerating simulated annealing for the permanent and combinatorial counting problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 900–907, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109656.
2. Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 143–152, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2767386.2767414.
3. Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Conference on Distributed Computing*, DISC'12, pages 195–209, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-33651-5_14.
4. Weiming Feng, Thomas P. Hayes, and Yitong Yin. Distributed symmetry breaking in sampling (optimal distributed randomly coloring with fewer colors), 2018. arXiv:1802.06953.
5. Weiming Feng, Yuxin Sun, and Yitong Yin. What can be sampled locally? In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 121–130, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3087801.3087815.
6. Manuela Fischer and Mohsen Ghaffari. A Simple Parallel and Distributed Sampling Technique: Local Glauber Dynamics. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing (DISC 2018)*, volume 121 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:11, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2018.26.
7. Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Struct. Algorithms*, 7(2):157–165, September 1995. doi:10.1002/RSA.3240070205.
8. Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.
9. Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 42–50, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2484239.2501983.
10. Hongyang Liu, Yitong Yin, and Yiyao Zhang. Work-efficient parallel counting via sampling, 2024. doi:10.48550/arXiv.2408.09719.
11. Sriram V. Pemmaraju and Joshua Z. Sobel. Exact distributed sampling. In *Structural Information and Communication Complexity: 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6–9, 2023, Proceedings*, pages 558–575, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-32733-9_25.
12. L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.
13. Daniel Štefankovič, Santosh Vempala, and Eric Vigoda. Adaptive simulated annealing: A near-optimal connection between sampling and counting. *J. ACM*, 56(3), May 2009. doi:10.1145/1516512.1516520.
14. Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *ACM-SIAM Symposium on Discrete Algorithms*, 2023. URL: https://api.semanticscholar.org/CorpusID:259937341.

Brief Announcement: Single-Round Broadcast: Impossibility, Feasibility, and More

Zhelei Zhou
Zhejiang University, Hangzhou, China

Bingsheng Zhang[1]
Zhejiang University, Hangzhou, China

Hong-Sheng Zhou
Virginia Commonwealth University, Richmond, VA, USA

Kui Ren
Zhejiang University, Hangzhou, China

Abstract

Broadcast is a fundamental primitive that plays an important role in secure Multi-Party Computation (MPC) area. In this work, we revisit the broadcast with selective abort (hereafter, short for broadcast) proposed by Goldwasser and Lindell (DISC 2002; JoC 2005) and study the *round complexity* of broadcast under different setup assumptions. Our findings are summarized as follows:

- We formally prove that 1-round broadcast is impossible under various widely-used setup assumptions (e.g., plain model, random oracle model, and common reference string model, etc.), even if we consider the static security and the stand-alone framework. More concretely, we formalize a notion called *consistent oracle* to capture these setups, and prove that our impossibility holds under the consistent oracle. Our impossibility holds in both honest majority setting and dishonest majority setting.
- We show that 1-round broadcast protocol is possible in the Universal Composition (UC) framework, by assuming stateful trusted hardwares. Our protocol can be proven secure against all-but-one adaptive and malicious corruptions. We bypass our impossibility result since our stateful trusted hardwares do not satisfy the definition of consistent oracle.
- We provide an application of 1-round broadcast: we construct the first 1-round multiple-verifier zero-knowledge (which is a special case of MPC) protocol, without assuming the broadcast hybrid world.

2012 ACM Subject Classification Security and privacy → Cryptography; Computing methodologies → Distributed algorithms

Keywords and phrases Broadcast, Security with abort, Round optimality

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.66

Funding *Bingsheng Zhang*: Supported by the National Natural Science Foundation of China (Grant No. 62232002) and Input Output (iohk.io).
Hong-Sheng Zhou: Supported in part by NSF grant CNS-1801470 and a VCU Research Quest grant.

1 Introduction

Broadcast [23] is an important and widely-used cryptographic primitive. Assume there are n parties $P_1, P_2, \ldots, P_n$. Broadcast allows one sending party (say, P_1) to send a message to the rest receiving parties (say, $P_2, \ldots, P_n$), and the receiving parties should receive the identical messages. Serving as a vital communication channel, broadcast plays an important role in the secure Multi-Party Computation (MPC) area [26, 16], for example, the classic

[1] Bingsheng Zhang and Hong-Sheng Zhou are the corresponding authors.

© Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren;
licensed under Creative Commons License CC-BY 4.0

MPC protocol [25] by Rabin and Ben-Or assumes that all parties have access to a broadcast channel. Furthermore, a series of round-optimal MPC protocols [6, 1, 3, 11, 20, 12, 21] are constructed in the broadcast hybrid world.

Round complexity of broadcast. In this work, we focus on the broadcast itself and study how to realize broadcast via only secure point-to-point (P2P) channels. More concretely, we consider *broadcast with selective abort* (hereafter, short for broadcast) proposed by Goldwasser and Lindell [17, 18]: the sending party holds a private input x and sends x to the receiving parties, and each honest receiving party either outputs x or a special symbol $\perp$ indicating abort. We study the *round complexity* of such broadcast protocols under different setup assumptions. Notice that, in this work, we do not consider Byzantine broadcast where the honest receiving parties are guaranteed to receive the same messages.

First of all, we clarify the meaning of "round". In this work, we consider simultaneous communication model, i.e., the parties are allowed to exchange messages in the same round; however, their messages should not depend on each other. For example, in the case of 1-round oblivious transfer protocol [4], the sender and the receiver can send their messages to each other simultaneously and their messages have no dependency; thus, it does not matter which party sends its message first.

Now we introduce the well-known broadcast protocol [17, 18] by Goldwasser and Lindell (hereafter, GL protocol), which is constructed in the plain model and is proven secure against all-but-one malicious corruptions. GL protocol is two-round: in the first round, the sending party sends the message m to the receiving parties; in the second round, the receiving parties send the received message to each other to check if they have received the same message. Since the receiving parties will echo-broadcast the received message from the sending party, GL protocol is also known as the echo-broadcast protocol. GL protocol and its variants are widely used in MPC protocols that are secure with abort.

Intuitively, it seems impossible to construct 1-round broadcast protocol in plain model; to the best of our knowledge, this impossibility result has not yet been formally proven in the literature. In addition to the plain model, there are some commonly-used setup assumptions, e.g., Random Oracle (RO) model, Common Reference String (CRS) model, and ideal cipher model, etc. We wonder if it is possible to construct 1-round broadcast protocol under these setup assumptions. This motivates our main research questions:

> *Is it possible to construct a 1-round broadcast protocol? If so, under what setup assumptions can such a protocol be constructed?*

1.1 Our Results

We investigate the above research questions, and our results are summarized as follows.

Impossibility: 1-round broadcast is impossible under most commonly used setups. We formally prove that 1-round broadcast protocol is impossible under many commonly-used setup assumptions (e.g., RO model, CRS model, ideal cipher model, preprocessing model, etc.). In the main body of this paper, we focus on the Universal Composition (UC) framework [9] and prove the impossibility result under UC framework. Note that, our impossibility result can be extended into the *stand-alone framework* [8, 15].

Feasibility: 1-round broadcast is possible under trusted hardwares. We provide a 1-round broadcast protocol in the *stateful* trusted hardware model, and it is proven to be UC-secure against an adaptive and rushing adversary who can corrupt up to $t < n$ parties. Furthermore,

our broadcast protocol is *non-interactive*: The only communication occurs when the sending party sends the messages to the receiving parties, and the receiving parties do not have to exchange the messages.

Application: 1-round multiple-verifier zero-knowledge. Finally, we present an application of our 1-round broadcast protocol: we construct the first 1-round Multiple-Verifier Zero-Knowledge (MVZK) [7] (a special case of MPC) protocol against a dishonest majority under trusted hardware via only secure P2P channels. Note that, there are three works [24, 2, 27] that claims to achieve 1-round online communication in the preprocessing model; however, they all assume their protocols are in the broadcast hybrid world. Due to the space limit, we leave the details of our MVZK protocol to the full version.

1.2 Related Work

There are limited works that consider broadcast with selective abort. As mentioned in Introduction, Goldwasser and Lindell proposed a 2-round broadcast protocol that is secure with selective abort in the plain model [17, 18] (hereafter, GL protocol). GL protocol is widely-used in MPC with abort (e.g. [13, 5]). In these MPC works, the authors are able to improve the communication complexity of GL protocol by introducing hash function [13] or utilizing the random linear combination technique [5]; however, to the best of knowledge, none of the previous works are able to reduce the round complexity of GL protocol.

2 Preliminaries

2.1 Notation

We denote the security parameter by $\lambda \in \mathbb{N}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible* if for every polynomial $p(\cdot)$ there exists λ_0 such that for all $\lambda > \lambda_0$ we have $\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}$. We write PPT for a probabilistic polynomial-time algorithm.

2.2 Digital Signature

The digital signature scheme can ensure the integrity of a message. A digital signature scheme (DS) consists of the following algorithms:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$: It takes the security parameter λ as input, and it outputs the public key pk and the signing key sk.
- $\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, m)$: It takes the signing key sk and the message m as inputs, and it outputs the signature σ for the message m.
- $\{0,1\} \leftarrow \mathsf{DS.Verify}(\mathsf{pk}, m, \sigma)$: It takes the public key pk, the message m and the signature σ as inputs, and it outputs a bit $b \in \{0,1\}$ indicating the acceptance or rejection.

We require the digital signature scheme to be correct and existentially unforgeable under chosen message attacks (EUF-CMA). Formally, we have the following definitions.

▶ **Definition 1** (Correctness). *We say a digital signature* $\mathsf{DS} = (\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *is correct if the following equation holds:*

$$\Pr\left[\begin{matrix}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda); \\ \sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, m)\end{matrix} : \mathsf{DS.Verify}(\mathsf{pk}, m, \sigma) = 1\right] = 1 \ .$$

▶ **Definition 2** (EUF-CMA Security). *We say a digital signature scheme* $\mathsf{DS} = (\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *is EUF-CMA secure if for any PPT adversary* $\mathcal{A}$*, the following equation holds:*

$$\Pr\left[\begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda);\\ \mathcal{L}_{\mathsf{Sign}} := \emptyset; (m^*,\sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})\end{array} : \begin{array}{c}\mathsf{DS.Verify}(\mathsf{pk},m^*,\sigma^*)=1\\ \wedge\, m^* \notin \mathcal{L}_{\mathsf{Sign}}\end{array}\right] \leq \mathsf{negl}(\lambda) \ ,$$

where $\mathcal{O}_{\mathsf{Sign}}(m)$ *will return* $\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, m)$ *and update* $\mathcal{L}_{\mathsf{Sign}} := \mathcal{L}_{\mathsf{Sign}} \cup \{m\}$.

2.3 Security Model

In this work, we construct protocols and analyze their security in the Universal Composition (UC) framework by Canetti [9]. We refer readers to see details in [9].

Secure communication model. We consider simultaneous communication [22]: the parties can exchange the messages in the same round, and the parties' messages should have no dependency. We also assume the parties are connected with only secure P2P channels.

Adversarial model. We consider both static and adaptive corruption. We consider a malicious and rushing adversary: the adversary can deviate from the protocol instruction, and it can delay sending messages on behalf of corrupted parties in a given round until the messages sent by all the honest parties in that round have been received. We assume that the adversary is allowed to corrupt the sending party and up to $t < n$ receiving parties, where n is the number of total receiving parties.

2.4 Broadcast Functionality

In this subsection, we provide the ideal functionality for broadcast $\mathcal{F}_{\mathsf{BC}}$ in Figure 1, which is adapted from [18]. The functionality $\mathcal{F}_{\mathsf{BC}}$ interacts with a set of parties $P_1, \ldots, P_n$ and the ideal-world adversary $\mathcal{S}$. Upon receiving the message m from any party, $\mathcal{F}_{\mathsf{BC}}$ will forward the message m to the rest receiving parties; however, $\mathcal{S}$ can cause any honest receiving party to abort. Notice that, in $\mathcal{F}_{\mathsf{BC}}$, $\mathcal{S}$ can make honest receiving parties abort, but $\mathcal{S}$ cannot make honest receiving parties output two distinct messages.

Functionality $\mathcal{F}_{\mathsf{BC}}$

The functionality interacts with a set of parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. Let $\mathcal{H}$ be the set of honest parties.

Upon receiving $(\textsc{Broadcast}, \mathsf{sid}, m)$ from any party P_i, do:

- Send $(\textsc{Broadcast}, \mathsf{sid}, P_i, m)$ to the adversary $\mathcal{S}$ and the corrupted parties.
- For each honest party $h \in \mathcal{H}$, wait for an input from the adversary $\mathcal{S}$, then do the following:
  - If it is $(\textsc{Continue}, \mathsf{sid}, h)$, send $(\textsc{Broadcast}, \mathsf{sid}, P_i, m)$ to the honest party h.
  - If it is $(\textsc{Abort}, \mathsf{sid}, h)$, send $(\textsc{Abort}, \mathsf{sid})$ to the honest party h.

Figure 1 The Ideal Functionality $\mathcal{F}_{\mathsf{BC}}$.

3 Impossibility under Various Setups

In this section, we prove that there exists no 1-round UC-secure broadcast protocol using only *consistent oracles* (to be defined later).

3.1 Consistent Oracles

We introduce the notion of a *consistent oracle* (Definition 3) to model a variety of global-setup behaviours. This notion is adapted from the *monotonically consistent oracle* of [14], which was originally used to establish impossibility results for non-interactive commitment and zero-knowledge protocols in global setups (e.g., the global random oracle) within the generalized UC framework [10].

▶ **Definition 3** (Consistent Oracles). *We say that an "oracle" (or Interactive Turing Machine) is* consistent *if it always returns the same response to the same query made by the same party P in one protocol session.*

▶ Remark 4. The consistent oracle captures a wide range of standard setup assumptions, e.g., the RO model, the CRS model, the PKI model, the preprocessing model, stateless trusted hardware (e.g., signature cards [19]), and combinations thereof.

3.2 Impossibility in the UC Framework

We prove that 1-round one-time broadcast is impossible to achieve using only consistent oracles, even in the presence of a static adversary. We use the method of proof by contradiction to prove the impossibility result. First of all, we assume there exists a 1-round one-time broadcast protocol using only secure P2P channels. Notice that, since we consider the simultaneous communication model, the sending party (say, P_1) is allowed to send messages to the receiving parties (say, $P_2, \ldots, P_n$) via secure P2P channels, and the receiving parties are allowed to communicate with each other, and each parties' messages should not depend on each other. Let us consider the case where the sending party P_1 gets corrupted. Let m, m' be two distinct messages, and let smsg_i (resp. smsg'_i) be the message that an honest sending party should send to the i-th honest receiving party P_i on input m (resp. m'); note that, the message smsg_i (resp. smsg'_i) may contain m (resp. m'). Let $\mathsf{rmsg}_{i,j}$ be the message that an honest receiving party should send to the j-th honest receiving party P_j. Upon receiving smsg_i (resp. smsg'_i) and $\{\mathsf{rmsg}_{i,j}\}_{i\in[2,n]\setminus\{j\}}$, the j-th honest receiving party P_j should output m (resp. m'). With above notations, we can consider the following adversary's strategy: the adversary can instruct the corrupted sending party to query the consistent oracle to prepare $\{\mathsf{smsg}_i\}_{i\in[2,n]}$ and $\{\mathsf{smsg}'_i\}_{i\in[2,n]}$; notice that, the corrupted sending party is capable of doing this by definition of the consistent oracle. Then the adversary can instruct the corrupted sending party to send smsg_2 to the first receiving party P_2, and send $\{\mathsf{smsg}'_j\}_{j\in[3,n]}$ to the rest receiving parties respectively; as a result, P_2 will output m while other receiving parties will output m', which violates the *consensus* property of the one-time broadcast protocol.

Next, we provide the formal theorem statement and the proof is deferred in the full version.

▶ **Theorem 5.** *Let $\mathcal{O}$ denote any PPT consistent oracle as defined in Definition 3. Let n be the number of total parties such that $n \geq 3$. There exists no 1-round one-time broadcast protocol that UC-realizes $\mathcal{F}_{\mathsf{BC}}$ depicted in Figure 1 in the $\mathcal{O}$-hybrid world using only secure P2P channels, in the presence of a static, malicious and rushing adversary who is allowed to corrupt the sending party.*

4 Feasibility under Trusted Hardwares

In this section, we show how to construct 1-round broadcast protocol in the stateful trusted hardware model; notice that, our stateful trusted hardware model does not satisfy the definition of consistent oracle, so we can bypass our impossibility result.

Now we introduce a new trusted hardware model, and we call it *counter-mode* trusted hardware. We also formally define it through an ideal functionality $\mathcal{O}_{\mathsf{HW}}$, which is put in Figure 2. We put our broadcast protocol $\Pi_{\mathsf{BC\text{-}HW}}$ in Figure 3, and state the security of $\Pi_{\mathsf{BC\text{-}HW}}$ through Theorem 6. The proof is deferred in the full version.

Functionality $\mathcal{O}_{\mathsf{HW}}$

It interacts with a set of parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. It is parameterized with $\mathsf{DS} := (\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$. It maintains a counter c, which is initialized as 0.

Initialize. Upon receiving $(\textsc{Init}, \mathsf{sid})$ from any party P_i, do:

- Generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.KeyGen}(1^\lambda)$ and record $(\mathsf{sid}, \mathsf{pk}, \mathsf{sk})$.
- Ignore any subsequent $\textsc{Init}$ command.

Get public-key. Upon receiving $(\textsc{GetPK}, \mathsf{sid})$ from any party P_i, do:

- If there is a $(\mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ has been recorded, return $(\textsc{GetPK}, \mathsf{sid}, \mathsf{pk})$ to the requester P_i.

Get signature. Upon receiving $(\textsc{Sign}, \mathsf{sid}, m)$ from any party P_i, if there is a $(\mathsf{sid}, \mathsf{pk}, \mathsf{sk})$ has been recorded, generate $\sigma \leftarrow \mathsf{DS.Sign}(\mathsf{sk}, (\mathsf{c}, m))$, return $(\textsc{Sign}, \mathsf{sid}, \sigma)$ to P_i, and increase $\mathsf{c} := \mathsf{c} + 1$.

Figure 2 The Functionality $\mathcal{O}_{\mathsf{HW}}$.

Protocol $\Pi_{\mathsf{BC\text{-}HW}}$

Input: P_i privately holds a message x.

Protocol:

- *(Initial phase)*: The parties send $(\textsc{Init}, \mathsf{sid})$ to $\mathcal{O}_{\mathsf{HW}}$. Each party P_j initializes a counter $\mathsf{c}_j := 0$ for $j \in [n]$.
- *(P_i broadcasts the message)*: P_i sends $(\textsc{Sign}, \mathsf{sid}, m)$ to $\mathcal{O}_{\mathsf{HW}}$, which returns $(\textsc{Sign}, \mathsf{sid}, \sigma)$. Then P_i increases its counter $\mathsf{c}_i := \mathsf{c}_i + 1$ and sends (m, σ) to the rest parties via secure P2P channels.
- *(P_j locally checks the validity of the received message)*: For each $j \in [n] \setminus \{i\}$, P_j sends $(\textsc{GetPK}, \mathsf{sid})$ to $\mathcal{O}_{\mathsf{HW}}$, which returns $(\textsc{GetPK}, \mathsf{sid}, \mathsf{pk})$. Then P_j checks if $\mathsf{DS.Verify}(\mathsf{pk}, (\mathsf{c}_j, m), \sigma) = 1$ holds. If so, P_j accepts m as the received broadcast message and increases its counter $\mathsf{c}_j := \mathsf{c}_j + 1$; otherwise, P_j aborts.

Figure 3 1-round broadcast protocol $\Pi_{\mathsf{BC\text{-}HW}}$ in the $\mathcal{O}_{\mathsf{HW}}$-hybrid world .

▶ **Theorem 6.** *Assume* $\mathsf{DS} = (\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Verify})$ *is a digital signature scheme that is correct and EUF-CMA secure. The protocol* $\Pi_{\mathsf{BC\text{-}HW}}$ *depicted in Figure 3 UC-realizes the functionality* $\mathcal{F}_{\mathsf{BC}}$ *depicted in Figure 1 in the* $\mathcal{O}_{\mathsf{HW}}$*-hybrid world, in the presence of an adaptive and malicious adversary who can corrupt up to* $t < n$ *parties.*

References

1 Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO 2017*.

2 Benny Applebaum, Eliran Kachlon, and Arpita Patra. Verifiable relation sharing and multi-verifier zero-knowledge in two rounds: Trading NIZKs with honest majority - (extended abstract). In *CRYPTO 2022*.

3 Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *CRYPTO 2018*.

4 Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *CRYPTO 1989*.

5 Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. In *CRYPTO 2021*.

6 Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *TCC 2017*.

7 Mike Burmester and Yvo Desmedt. Broadcast interactive proofs (extended abstract). In *EUROCRYPT 1991*.

8 Ran Canetti. Security and composition of multiparty cryptographic protocols. In *Journal of Cryptology 2000*.

9 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*.

10 Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC 2007*.

11 Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In *TCC 2020*.

12 Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In *EUROCRYPT 2022*.

13 Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*.

14 Yevgeniy Dodis, Victor Shoup, and Shabsi Walfish. Efficient constructions of composable commitments and zero-knowledge proofs. In *CRYPTO 2008*.

15 Oded Goldreich. Foundations of cryptography, volume 2.

16 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC 1987*.

17 Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC 2002*.

18 Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. In *Journal of Cryptology 2005*.

19 Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *Central European Conference on Cryptology 2005*.

20 Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. On the round complexity of black-box secure MPC. In *CRYPTO 2021*.

21 Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box MPC in the plain model. In *CRYPTO 2023*.

22 Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC 2013*.

23 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. In ACM Trans. Program. Lang. Syst. 1982.

24 Matt Lepinski, Silvio Micali, and Abhi Shelat. Fair-zero knowledge. In *TCC 2005*.

25 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC 1989*.

26 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS 1982*.

27 Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. Single-input functionality against a dishonest majority: Practical and round-optimal. In *PKC 2025*.