4th International Conference on Blockchain Economics, Security and Protocols

Tokenomics 2022, December 12–13, 2022, Paris, France

Edited by

Yackolley Amoussou-Guenou
Aggelos Kiayias
Marianne Verdier

OASICS

Editors

Yackolley Amoussou-Guenou
Université Paris-Panthéon-Assas, CRED, Paris, France
Yackolley.Amoussou-Guenou@u-paris2.fr

Aggelos Kiayias
University of Edinburgh, UK
IOG, Edinburgh, UK
Aggelos.Kiayias@ed.ac.uk

Marianne Verdier
Université Paris-Panthéon-Assas, CRED, Paris, France
marianne.verdier@u-paris2.fr

ACM Classification 2012
Theory of computation → Distributed algorithms; Security and privacy → Cryptography; Mathematics of computing → Mathematical analysis

ISBN 978-3-95977-274-7

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/978-3-95977-274-7.

Publication date
July, 2023

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

Digital Object Identifier: 10.4230/OASIcs.Tokenomics.2022.0

OASIcs – OpenAccess Series in Informatics

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface

This volume includes the published papers of Tokenomics 2022, the fourth edition of the International Conference on Blockchain Economics, Security and Protocols which took place on December 12th – 13th 2022 and was hosted at Sorbonne Université, Paris, France.

Tokenomics is an international forum for theory, design, analysis, implementation and applications of blockchains and smart contracts. The goal of the conference is to bring together economists, computer science researchers and practitioners working on blockchains in a unique program featuring outstanding invited talks and academic presentations.

Following the tradition since its very beginning, the Program Committee of the conference was divided into two complementary sub-committees. The *Distributed Computing* sub-committee consisted of 23 expert computer science researchers, chaired by Aggelos Kiayias, and the *Economics* sub-committee consisted of 28 expert economics researchers, and was chaired by Marianne Verdier, with Yackolley Amoussou-Guenou as co-chair.

In total there were 55 submissions for consideration 11 mainly targeted to the distributed computing sub-committee and 44 mainly targeted toward the economics sub-committee. On average, submissions toward the distributed computing sub-committee got three reviews each, and submissions to the economics sub-committee got two reviews each. After the collection of all the reviews from the program committee members, the program committee chairs met to choose the accepted papers for the conference, and, given the reviews, took into account some constraints such as the topic of the papers. We would like to mention that the selection was done irrespective to which sub-committee they were submitted to. The objective was to have presentations fitting the goals of the conference and create interdisciplinary and coherent sessions. The program committee deliberations concluded with 16 papers being accepted for the final program, ranging from studies of consensus protocols to distributed applications. We give below the list of the contributions presented by session.

- MEV and front-running. This session grouped papers analyzing the problem of front-running attacks in distributed applications. In particular the papers studied different ways to mitigate them. It consisted of the following presentations: "*Credible Decentralized Exchange Design via Verifiable Sequencing Rules*" by Ferreira and Parkes; "*The Evolution of Blockchain: From Public to Private Mempools*" by Jia, Capponi and Wang; and "*Commitment Against Front Running Attacks*" by Canidio and Danos.
- DeFi. In this session the presentations analyzed different aspects of decentralised finance and possible ways to improve them. It consisted of the following presentations: "*The Need for Fees at a DEX: How Increases in Fees Can Increase DEX Trading Volume*" by Hasbrouck, Rivera, and Saleh; "*Interest Rate Parity in Decentralized Finance*" by Chaudhary, Kozhan, and Viswanath-Natraj; and "*Token Incentives and Platform Competition: a Tale of two swaps*" by Liu, Chen, and Zhu.
- Blockchain, CBDC and Payments. This session covered various presentations related to central bank digital currencies (CBDC), payments platforms, and user adoption of innovative payment instruments. It consisted of the following presentations: "*The Demand for Programmable Payments*" by Kahn and van Oort; "*CBDC and Payment Platform Competition*" by Liu, Reshidi, and Rivadeneyra; and "*Central Bank Digital Currency and Banking Choice: The Impact of Service Location*" by Usher, Li, and Zhu.
- Smart contracts, oracles and AMMs. The presentations of this session were about smart contracts. In particular, some works studied the functioning of automated market makers from a theoretical perspective, and the last presentation exposed why truth-telling cannot

always be guaranteed with oracles. It consisted of the following presentations: "*Axioms for Constant Function AMMs*" by Schlegel and Mamageishvili; "*Consistency of automated market makers*" by Danos and Wang; and "*An Impossibility Theorem on Truth-Telling in Fully Decentralized Systems*" by Garratt and Monnet.

- Efficiency of Blockchain protocols. This session focused on improving the efficiency and the analytical tools for blockchain (or more broadly, consensus) protocols. It consisted of the following presentations: "*Equilibrium Staking Levels in a Proof-of-Stake Blockchain*" by John, Rivera, and Saleh; "*Maximal Extractable Value (MEV) Protection on a DAG*" by Malkhi and Szalachowski; "*An Economic Model of Consensus on Distributed Ledgers*" by Halaburda, He, and Li; and "*QPQ 1DLT: A System For the Rapid Deployment of Secure and Efficient EVM-Based Blockchains*" by Bottoni, Datta, Franzoni, Ragnoli, Ripamonti, Rondanini, Sagirlar, and Trombetta.

At the end of the conference, on the basis of an assessment of the content of the article as well as the presentation, Charles Kahn and Maarten van Oort received the best-paper prize for their paper on "*The Demand for Programmable Payments*", thanks to the sponsoring of Keyrock.[1]

In addition to the accepted papers, the program included three stimulating keynote presentations. The topics were chosen after discussing with the speakers about three important directions for future research: the regulation of cryptoassets, the role of tokens in shaping firms' incentives to adopt a platform business model, and the analysis of the players' strategies via the use of algorithmic game theory. Claudine Hurman (Banque de France, France), director of Infrastructures, Innovation and Payments Regulating crypto-assets and experimenting at the French central bank presented "*CBDC: two sides of the same coin*"; Hanna Halaburda (New York University, Stern School of Business, USA) presented "*How Blockchain Tokens are Changing Platform Economics*"; and Elias Koutsoupias (Oxford University, UK), 2012 Gödel prize, presented "*Algorithmic game theory and blockchains*".

Overall, the topical works presented at the conference, as well as the numerous interactions (also between different disciplines), make us think that the conference was a success. We hope this venue will continue fostering interactions between economists, computer science researchers and practitioners, and will further stimulate interdisciplinary research in the broader area.

We are grateful to the sponsors of this fourth edition of the Tokenomics conference: the Blockchain@Polytechnique chair, the Finance Digitale chair, Keyrock, IOG, and the LIP6, Sorbonne Université.

We would like to thank the authors for submitting their work to the conference and the program committee members who worked very hard in reviewing papers and giving feedback to the authors. We are grateful for their help in building this great program. A special thanks to all the presenters for their well prepared and clear presentations, and to the audience for all the discussions that took place. We also thank all the volunteers who helped the conference run smoothly.

Last but not least, we would like to thank Julien Prat and Maria Potop-Butucaru, the general co-chairs of Tokenomics 2022 for their help and advice for the organization of the conference.

Aggelos, Marianne, and Yackolley

[1] Keyrock (`https://keyrock.eu/`) sponsored the best paper award, as part of having industrial practitioners at the conference, one of the conference's goals.

List of Authors

Simone Bottoni (3)
RTM, Lugano, Switzerland

Andrea Canidio (7)
IMT School for Advanced Studies, Lucca, Italy;
CoW Protocol, Paris, France

Amit Chaudhary (5)
Warwick Business School, University of Warwick,
Coventry, UK

Vincent Danos (4, 7)
CNRS, Paris, France;
DI ENS, INRIA, PSL, Paris, France

Anwitaman Datta (3)
Nanyang Technological University, Singapore,
Singapore

Federico Franzoni (3)
Unaffiliated Researcher, Barcelona, Spain

Hanna Halaburda (2)
Stern School of Business, New York University,
NY, USA

Charles M. Kahn (8)
Department of Finance, University of Illinois,
Urbana-Champaign, IL, USA

Elias Koutsoupias (1)
University of Oxford, UK

Roman Kozhan (5)
Warwick Business School, University of Warwick,
Coventry, UK

Dahlia Malkhi (6)
Chainlink Labs, UK

Maarten R.C. van Oordt (8)
Tinbergen Institute, Amsterdam,
The Netherlands;
Vrije Universiteit Amsterdam, The Netherlands

Emanuele Ragnoli (3)
RTM, Lugano, Switzerland

Roberto Ripamonti (3)
RTM, Lugano, Switzerland

Christian Rondanini (3)
RTM, Lugano, Switzerland

Gokhan Sagirlar (3)
RTM, Lugano, Switzerland

Pawel Szalachowski (6)
Chainlink Labs, UK

Alberto Trombetta (3)
Insubria University, Varese, Italy

Ganesh Viswanath-Natraj (5)
Warwick Business School, University of Warwick,
Coventry, UK

Weijia Wang (4)
ENS, Paris, France

Program Committee

General Chairs

- Julien Prat, ENSAE, CREST, Ecole Polytechnique
- Maria Potop-Butucaru, LIP6, Sorbonne Université

Distributed Computing

- Aggelos Kiayias, University of Edinburgh and IOG (Chair)
- Elli Androulaki, IBM Research – Europe
- Vincent Danos, CNRS
- Ittay Eyal, Technion
- Matheus Xavier Ferreira, Harvard
- Juan Garay, Texas A&M University
- Arthur Gervais, Imperial College
- Maurice Herlihy, Brown University and Algorand
- Dimitris Karakostas, University of Edinburgh
- William J. Knottenbelt, Imperial College
- Philip Lazos, IOG
- Andrew Lewis-Pye, London School of Economics
- Francisco Marmolejo, Harvard
- Tal Rabin, University of Pennsylvania and Algorand Foundation
- Tim Roughgarden, Columbia University and a16z Crypto
- Jan Christoph Schlegel, City University of London
- David Siska, University of Edinburgh
- Qiang Tang, University of Sydney
- Vanessa Teague, Thinking Cybersecurity
- Sara Tucci-Piergiovanni, CEA-LIST, Université Paris-Saclay
- Catherine Tucker, MIT Chryssis Georgiou, University of Cyprus
- Dimitrios Vasilopoulos, IMDEA Software Institute
- Dionysis Zindros, Stanford

Economics

- Marianne Verdier, Université Paris-Panthéon-Assas (chair)
- Yackolley Amoussou-Guenou, Université Paris-Panthéon-Assas (co-chair)
- Arash Aloosh, NEOMA Business School
- Vlad Babich, Georgetown University
- Christophe Bisière, Toulouse School of Economics
- Matthieu Bouvard, Toulouse School of Economics
- Agostino Capponi, Columbia University
- Catherine Casamatta, Toulouse School of Economics
- Jonathan Chiu, Bank of Canada
- Will Cong, Cornell University
- Bertrand Crettez, Université Paris-Panthéon-Assas
- Michele Fabi, ENSAE, CREST, Ecole Polytechnique
- Rod Garratt, BIS
- Guillaume Haeringer, Baruch College

Algorithmic Game Theory and Blockchains

Elias Koutsoupias
University of Oxford, UK

Abstract

Algorithmic game theory has developed into a mature field over the past three decades. However, the emergence of blockchains has raised new fundamental questions at the intersection of computer science, economics, and game theory.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Blockchains, Mining games, Reward sharing schemes, Distributed game theory

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.1

Category Invited Talk

1 Algorithmic game theory

Incentives play an increasingly important role in computer science. It is now hard to imagine a time when game-theoretic issues were not part of computer science, but it was a mere three decades ago that incentives entered the computer science narrative.

To be clear, algorithmic issues of game theory questions have been considered since the inception of game theory in the 1940s. Similarly, game theorists have traditionally contributed significantly to the algorithmic theory. For example, the development of prediction algorithms in the 1950s was part of the game theory agenda, and algorithms for equilibria in the 1960s were great examples of successful algorithms.

About three decades ago there was a significant shift when it became clear that the traditional analysis of algorithms was insufficient. The reason was the widespread adoption of the internet and the web, whose protocols run at the participating nodes. It was realized that the users may have an incentive not to follow the prescribed use. One can argue that algorithmic game theory was born out of such considerations.

Research in the new field have been mainly concentrating around three research branches:

- Computational issues of game theory and economics. This research area focuses on the algorithmic aspects of game theory and economics. A significant breakthrough in this area was the proof that the computation of a Nash equilibrium of a game is PPAD-complete, which strongly suggests that the problem is intractable.
- Price of anarchy. This research area investigates the quality of equilibria. A major success in this area has been the discovery that congestion games have relatively small price of anarchy, meaning that even when players act selfishly, the outcome is close to the socially optimal one.
- Algorithmic mechanism design. This research branch studies how to design mechanisms that incentivize participants to behave in a desired way, even when they have conflicting interests. Some notable achievements in this field include the development of efficient algorithms for computing almost optimal mechanisms, and the resolution of the Nisan-Ronen conjecture, which had been an open problem for many years.

2 Incentives and blockchains

Many research directions are still active in traditional algorithmic game theory, but the advent of blockchains has changed the game. Blockchains have brought new fundamental questions at the intersection of computer science, games, and economics. Here are a few examples.

2.1 Mining games

An ideal blockchain is a sequence of blocks and the process of adding blocks to it is known as “mining.” In proof of work blockchains, users can add a block only by solving a crypto-puzzle, while proof of stake systems randomly select users with probability proportional to their stake. However, the distributed nature of the system means that changes are not immediately communicated to all users, which can result in multiple users extending the blockchain from the same point, creating a tree structure instead of a path. To address this, the protocol advises miners to create blocks at the end of the longest known branch and immediately propagate them to the network. If all miners follow this advice, the reward structure guarantees revenue proportional to computational power or stake. Nevertheless, selfish miners may deviate from the advice if it serves their interests, raising the question of which “mining game” they play and what equilibria exist in these games. Unlike games in classical game theory where players’ strategies and utilities are given explicitly or implicitly, mining games are indirectly defined by the blockchain protocol.

2.2 Reward sharing schemes

Blockchains require an adequate number of active participants to maintain the system. Blockchain consensus protocols incentivize every user to be an active participant by paying them for mining a new block. However, this has led to undesirable effects, such as excessive energy consumption and a concentration of power among a small number of mining pools. Reward sharing schemes attempt to address these issues by incentivizing the formation of pools of users in a transparent manner. To form a pool, users delegate their stake to one of its members, the pool leader, who runs the protocol on their behalf.

These schemes are based on payments that attempt to encourage a good selection of pool leaders. The quality of a reward scheme is determined by criteria such as liveness, efficiency, decentralization, and Sybil resiliency.

2.3 Distributed computing with incentives

In traditional mechanism design, a mediator collects inputs from participants and runs payment and allocation algorithms. However, in blockchains, this takes the form of smart contracts. Smart contracts are algorithms that have a state, which updates when appropriate transactions are issued by members. While there is a lot of hype surrounding the potential of smart contracts, there are also significant risks associated with them due to the complex game-theoretic analysis required even for simple contracts. Despite these challenges, the number of deployed smart contracts on actual blockchains is growing at a rapid pace, making it crucial to develop a robust theory to understand their power and limitations.

Smart contracts are fundamentally distributed algorithms with incentives. The theory of distributed computation has been studied extensively, but without incentives. A notable success in this area was the characterization of what can be computed by distributed protocols, which involved discovering a surprising connection to algebraic topology. The question now is whether a similar characterization exists for distributed tasks when participants act selfishly.

How Blockchain Tokens Are Changing Platform Economics

Hanna Halaburda
Stern School of Business, New York University, NY, USA

Abstract

Blockchain technologies are technologies inspired by Bitcoin, which emerged in 2008. Since then, many cryptocurrencies, altcoins, and other blockchain applications have emerged. For example, Ethereum introduced smart contracts, and with them came tokens, fungible tokens, non-fungible tokens, decentralized finance (DeFi), and decentralized autonomous organizations (DAOs). All these technologies can be grouped under the umbrella term "blockchain technologies."

Each new generation of blockchain technology promises decentralization, disintermediation, a level playing field for entry, and improved value creation and distribution. However, it is essential to examine to what extent and under what conditions blockchain technologies deliver on these promises. It turns out that sometimes they do, and sometimes they do not. This distinction is essential to apply blockchain technologies effectively for large-scale practical applications.

I focus here on blockchain-based cryptographic tokens and their impact on platform economics. Blockchain-based tokens, in conjunction with smart contracts, allow for new design choices in platforms. Therefore, I explore how these new design choices may help solve old problems in platform economics.

2012 ACM Subject Classification Applied computing → Digital cash; Applied computing → E-commerce infrastructure; Applied computing → Electronic funds transfer; Applied computing → Online banking; Social and professional topics → Consumer products policy

Keywords and phrases blockchain, token, platform economics

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.2

Category Invited Talk

1 Problems in Platform Economics

The main problems in platform economics come from network effects. Network effects occur when the value of a good or product increases as more people use it. Platforms create value by bringing users together. For example, social media platforms like Facebook bring people together to interact, while Uber brings different types of users, such as drivers and passengers. As more users join the platform, its value increases due to the network effects. This distinct dynamic is not found in other types of goods.

When a platform has no users, potential users are hesitant to join. They will only consider joining if they anticipate that others will also join. This dynamic creates a challenge for new platforms to enter the market with network effects. They must have a significant presence early on, or they risk being unable to enter the market. For example, starting Facebook with just a few users would be difficult. The platform needs a critical mass of users before it can expand.

Once a platform has a large-scale market, network effects create barriers to entry for competitors. These barriers allow the platform to enjoy considerable market power and extract value from the system. For example, Facebook can extract data from its users, while other platforms may charge high prices or impose burdensome conditions on users to extract value. For example, in the past, platforms have charged high prices for long-distance calls.

Currently, there is ongoing debate about Facebook's ability to extract data. The result of these network effects is the creation of powerful intermediaries that can extract value from the system.

I explore the potential of blockchain-enabled tokens to address these challenges of barriers to entry and value capture. Tokens have been around for a long time, but their use in the context of blockchain is relatively new. Blockchain-enabled tokens offer two key features that traditional tokens do not: commitment and traceability. Commitment refers to the fact that platforms can commit to specific uses, acceptances, or restrictions of tokens, while tradability means that tokens can be traded independently of the platform that issued them. This independence enables tokens to be listed on cryptocurrency exchanges, such as Helium or Filecoin, and traded freely.

I focus here on two types of tokens: utility tokens and governance tokens. Utility tokens enable the platform to issue tokens instead of charging prices directly. This approach can make entry into a market with network effects easier, as it lowers the barriers to entry for new players. However, we also find that utility tokens lower the overall platform's profit and decrease overall welfare, which may limit their usefulness in certain conditions.

Governance tokens, in turn, facilitate voting and thus enable the creation of decentralized autonomous organizations, which can help distribute value more equitably. We find that governance tokens can lead to more value creation and equitable value distribution, but their treatability may also lead to actual re-centralization in equilibrium and value capture by anonymous users. This outcome may be different or even worse than value capture by regulated parties, which can be brought to court.

My research indicates that blockchain-based tokens may help to mitigate the challenges of platform economics, although they may not eliminate them entirely. While tokens present exciting opportunities, it is essential to be aware of potential risks and limitations associated with their use. By understanding the conditions under which tokens can be beneficial, we can evaluate their potential for enhancing economic outcomes.

2 Utility tokens and barriers to entry

Let's start by discussing utility tokens and platform entry. The inspiration for our research comes from great examples like Filecoin and Helium. Filecoin is a peer-to-peer competitor to Dropbox for file storage. It is a peer-to-peer file storage system where users give access to their computer's space to store other people's files. Users need to be willing to pay for this storage, and they pay those who provide storage with Filecoin. Another example, Helium, is a peer-to-peer hotspot network designed for use with IoT devices. Its design relies on people allowing the system to use their hotspots to some capacity, and together they can connect large spaces and allow for connectivity. In both of these services, there are strong network effects. Thus, each system is only attractive if numerous users have already joined the system. The systems are not useful with just a few people offering storage or hotspot capacity.

Both Filecoin and Helium use tokens. Initially, Helium had difficulty getting off the ground and entering the market. However, later it started to use cryptographic tokens, and with those tokens, it successfully took off and is now an active network. In both cases, we have network effects and potential users want to join only if other users join as well. This dependency results in two equilibria, one where everybody adopts and another where nobody adopts. Such multiplicity of equilibria, in turn, leads to a coordination problem. This observation for markets with network effects has a long history going back to Katz and Shapiro (1985) [5] in the case of one-sided networks or Caillaud and Jullien (2001, 2003) [3, 4] and Rochet and Tirole (2003, 2006) [6, 7] in the case of two-sided platforms.

The traditional way to solve this coordination problem is through subsidy, also called a divide-and-conquer strategy. It means subsidizing early users and then profiting from those who join later when it is clear that the platform is successful. Since the platform can charge high prices later when it is successful, it is profitable to subsidize early users so that they join and try the platform. There are many examples of applications of that strategy. Two of my favorite examples are PayPal and Uber because they would give early users a ten-dollar credit for setting up an account and joining the network. It is a very successful strategy, but the problem is that it requires upfront capital, which may be expensive and create barriers to entry.

In *Overcoming the Coordination Problem in New Marketplaces via Cryptographic Tokens* (Bakos and Halaburda 2022 [1]), Yannis Bakos and I build a simple multi-period model analyzing whether blockchain-based utility tokens allow the platform to overcome the barriers to entry without the need for subsidy. Every period, new people arrive at the market and decide whether to join the platform. With the subsidy, the platform sets the access price every period, and the price may be negative. Typically the early users are subsidized. The people arriving at the market may not be fully aware of how much they will like the platform. They only learn that after they join. So if they try it, they may learn that they do not like the platform. If they leave the platform under the subsidy scheme, they get their outside option (which we normalize to 0).

But instead of subsidy, the platform may decide to issue utility tokens. A utility token may allow you to access the platform service, and conversely, you can access the service only with these tokens. This is the case with Helium or Filecoin platforms. The platform can set the price of the tokens in the first period. In the future, however, the platform only decides on the number of additional tokens it issues, and the market determines the price of the tokens by balancing the supply and demand because the tokens are tradable independently of the platform.

If, after experiencing the platform, a user wants to exit, he can sell his token. The price of those tokens is positive (in expectations). This is because even though the platform can issue an arbitrary number of additional tokens, it will never be optimal to issue so many that the price drops to 0 (barred some shock realization in an uncertain environment). Thus, the users expect that they will be able to sell the token at a positive price if they do not like the platform.

Thus, utility tokens may induce agents to try the platform without subsidy. They may even be willing to pay for the token, while without the token, they would need to be paid to join. So the platform's first-period profit is higher with tokens than under the subsidy, and tokens may eliminate the need for subsidy altogether.

However, the platform's future profit is lower with tokens for two reasons. First, the platform will sell fewer tokens in the future because some new users buy tokens from existing early users rather than from the platform directly. Second, the price at which the platform can sell the tokens in the future is lower than the price it could charge under the subsidy. This is because an additional supply of tokens from the existing users puts pressure on the price. But also, the future network is smaller, so less valuable for the new users. The network is smaller under tokens than under subsidy because the positive resale price of the token makes leaving more attractive for early users. Recall that under the subsidy, the leaving users get nothing, but with tokens, they can sell their tokens at a positive price. Hence, some early users who would stay on the platform and provide some network benefit under subsidy are enticed to leave with the token's positive resale price.

Note that the platform can shut down both forces leading to lower prices in the future by reneging on accepting old tokens or limiting the trading of tokens. And if the platform realizes higher first-period profit and successfully enters without the need for subsidy, it will have a strong incentive to intervene in this way to increase its future profits. But if agents expect such reneging, they would refrain from acquiring the tokens in the first place.

Blockchain and smart contracts play a crucial role in this result. With smart contracts, the platform can commit to accepting the tokens in the future. Moreover, the platform cannot interfere with trading because trading occurs on a censorship-resistant blockchain. These two features were not available with pre-blockchain tokens but are crucial to realizing the benefit of overcoming the barrier to entry arising from network effects.

With tokens, there are two competing forces affecting platform profit. The platform earns higher early profits with tokens than under subsidy and lower future profits. It turns out that the second effect is always more prominent, and the platforms' overall profit is lower under tokens. Furthermore, the overall welfare is lower. And this is again because more early users exit with tokens due to the positive resale price, and thus overall network will be smaller in the future, which means that overall network effects generated by the platforms are smaller. Moreover, a bigger part of the welfare goes to the users because they resell the tokens. So tokens redistribute welfare from the platform to the users, but not evenly. The users who exit benefit, but the users who stay are worse off because they don't get to participate in the larger network that would be there under the subsidy.

Overall, blockchain-based cryptographic tokens help overcome the coordination problem for new entrants, and they can make entry possible without a subsidy. They do so by moving the revenue from the future to the entry stage. However, the overall profit and welfare are smaller. So wherever subsidy is (cheaply) available, it would be preferable for the platform. Nonetheless, in many cases, especially for novel technologies, up-front capital needed for a subsidy may come at a high cost or even be unavailable altogether. In such a case, tokens provide a viable entry option. This may explain why we see Hellium and Filecoin successfully entering the market using utility tokens, while it is unlikely an attractive option for more traditional businesses such as Uber or Lyft.

Governance tokens and value distribution

The second type of tokens we considered are the governance tokens and their effect on value creation and distribution?these governance tokens power decentralized autonomous organizations (DAOs). Despite the spectacular failure of The Dao in 2016, many DAOs have been operating quite successfully, like MakerDao or UniSwap. DAOs are set up using smart contracts and governance tokens with the purpose of decentralizing decision-making by allowing a large number of people to vote with their governance tokens. Moreover, other smart contracts can automatically implement those decisions after the vote.

DAOs boast the ability to prevent powerful intermediaries from capturing the value. They aim to achieve this goal by decentralization, much like other blockchain technologies. The argument goes that instead of relying on intermediaries that charge higher prices due to network effects, DAOs offer decision-making power to voters who are also users of the platform. Thus, they will have no incentive to increase the prices and will not extract the value created by the growing network. Furthermore, DAOs typically operate in a permissionless environment to prevent authority figures from excluding users and capturing value. If an agent or entity has the power to exclude others from the DAO, they may extract payments for access.

In *Will Blockchains Disintermediate Platforms? The Problem of Credible Decentralization in DAOs* (Bakos and Halaburda 2023 [2]), Yannis Bakos and I discovered that DAOs could realize the promise of better value creation and distribution, but only under specific circumstances. However, under more general circumstances, DAOs experience strong forces returning to centralization that can hinder this outcome. Hence, it may be challenging or even impossible for platform intermediaries controlled by democratic DAOs to exist in the long run due to these re-centralization forces. Therefore, we suggest that additional measures are required to maintain these intermediaries.

Let me here illustrate our point in a simple example. Imagine there are n potential platform users, and everyone's benefit or "utility" from using the platform ranges from 0 to 1. Those who don't use the platform receive no benefit. This scenario results in a 45-degree demand curve, $1 - p$, often seen in Microeconomics 101 textbooks. Additionally, we will assume that there is no cost to providing the platform. In such a market, the most profitable price for a monopoly platform is $p_m = 1/2$, which generates a monopoly profit of $p_m(1 - p_m) = 1/4n$. However, this also creates a deadweight loss because half of the market cannot benefit from using the platform.

Now, let's explore an alternative approach where the price for accessing the platform is determined by the DAO as a collective decision. We focus here on the pricing decision because this is the most straightforward textbook decision and will easily demonstrate our point. But this approach could be applied to any value-extracting decision, such as selling data or allowing advertising on a platform. In this scenario, there is no central authority controlling pricing. Instead, all potential users have a governance token that allows them to vote on the price. Once the price is set, users can choose to pay it and join the platform or decline and not join. If the price is positive, the DAO earns a profit which is then distributed as a dividend to token holders.

Such a DAO can vote for a price of zero. At this price, everybody would participate in the platform and benefit from its utility, leading to maximum social welfare without any deadweight loss. Next, we examine under what conditions the DAO will ultimately set this price.

First, let's notice that some potential users who do not find much value in joining the platform may prefer to set a higher price and earn more of a dividend rather than setting a lower price and participating in the platform. However, the cost of joining outweighs the dividend for those who plan on using the platform, so they prefer to set the price at zero. Each token can bring a maximum dividend of $\frac{p_m(1-p_m)}{n} = 1/4$ when the monopoly price is set; therefore, those who value joining the platform at less than $1/4$ will vote for the monopoly price, while those who value it higher will vote for zero price. In our example, $3/4$ of token holders vote for the zero price, resulting in the DAO setting the socially-optimal price through a simple majority rule. This is a very encouraging result. Note, however, that until now, we have assumed that all token holders hold at most one token.

It turns out that it is no longer guaranteed that the DAO would set zero access price when users hold multiple tokens. A user with multiple tokens prefers setting a higher access price as his token holdings increase. To see that, note that the net benefit of a user owning t tokens with the access price p_{DAO} is

$$\underbrace{\theta}_{\substack{\in [0,1] \\ \text{usage utility}}} - \underbrace{p_{DAO}}_{\substack{\text{price} \\ \text{(user pays)}}} + t \underbrace{p_{DAO}\,(1 - p_{DAO})}_{\substack{\text{dividend} \\ \text{(others pay)}}} .$$

The access price that would maximize the net benefit for a user owning t tokens is $p^*(t) = \frac{t-1}{2t}$, which increases with the user's token holdings. Thus, the more tokens the user holds, the higher price he would prefer to set. For example, even with just two tokens, a user would rather set a price of 1/4 than a zero price. This is because, with more tokens, the dividend contributes more to the user's payoff, making a larger dividend more valuable relative to the cost of paying for access themselves.

Moreover, tradeable tokens can lead to a significant level of concentration. If the access price is positive, the governance token will have a positive value due to the expected dividend. As a result, users may choose to purchase more tokens to receive more dividends. However, purchasing a majority of tokens will yield even greater benefits, as it allows the user to influence the price increase, further raising their dividend earnings. We have found that in equilibrium one user will purchase enough tokens to gain a majority. This is another encouraging outcome because it means that although re-centralization may occur, there will be a limit to concentration.

However, this limit to concentration may have little effect on the value capture. Note that if one user holds $\frac{n+1}{2}$ tokens, the dao will set $p_{DAO} = \frac{1}{2} - \frac{1}{n+1}$, which for large n is very close to 1/2, the monopoly price. As a result, the value is indeed more distributed, as the holders of the remaining tokens also receive high dividends, but the efficiency of the market is only marginally improved.

How can such dynamics be prevented? One possibility is to limit the number of governance tokens that a single person can acquire, but this can be challenging in a permissionless environment where one individual may control multiple wallets. Another option is to restrict or disable the trading of tokens, as seen in the Soulboud token proposal (Weyl, Ohlhaver and Buterin 2022 [8]). However, many DAOs value the transferability of governance tokens and have not yet implemented such restrictions.

3 Conclusions

Blockchain-based cryptographic tokens, like utility and governance tokens, offer new solutions to old problems in platform economics. But they may come at a price or not work as expected. As we have shown, utility tokens may improve the viability of a new platform, but at the cost of its overall profits. So it means that there is still room for banks and VCs alongside the tokens.

The governance tokens may allow for decentralization and an increase in welfare, but they do not remove the tendency to re-centralize and extract surplus. Notice that when the Internet emerged, it also promised decentralization, disintermediation, and democratization. But after the initial years of decentralization, the Internet enabled the rise of intermediaries with more power than ever before – Google, Amazon, or Facebook. As we are looking toward blockchain and DAOs for the new wave of decentralization, it is possible that without changes in design or additional safeguards, we may end up on the same trajectory leading to a new generation of powerful intermediaries.

References

1 Yannis Bakos and Hanna Halaburda. Overcoming the coordination problem in new marketplaces via cryptographic tokens. *Info. Sys. Research*, 33(4):1368–1385, December 2022. doi:10.1287/isre.2022.1157.

2 Yannis Bakos and Hanna Halaburda. Will blockchains disintermediate platforms? The problem of credible decentralization in daos. *SSRN*, April 8, 2023. doi:10.2139/ssrn.4221512.

3 Bernard Caillaud and Bruno Jullien. Competing cybermediaries. *European Economic Review*, 45(4):797–808, 2001. 15th Annual Congress of the European Economic Association. doi:10.1016/S0014-2921(01)00123-4.

4 Bernard Caillaud and Bruno Jullien. Chicken & egg: Competition among intermediation service providers. *The RAND Journal of Economics*, 34(2):309–328, 2003. URL: `http://www.jstor.org/stable/1593720`.

5 Michael L. Katz and Carl Shapiro. Network externalities, competition, and compatibility. *The American Economic Review*, 75(3):424–440, 1985. URL: `http://www.jstor.org/stable/1814809`.

6 Jean-Charles Rochet and Jean Tirole. Platform Competition in Two-Sided Markets. *Journal of the European Economic Association*, 1(4):990–1029, June 2003. `doi:10.1162/154247603322493212`.

7 Jean-Charles Rochet and Jean Tirole. Two-sided markets: A progress report. *The RAND Journal of Economics*, 37(3):645–667, 2006. URL: `http://www.jstor.org/stable/25046265`.

8 E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized society: Finding web3's soul. *SSRN*, May 10, 2022. `doi:10.2139/ssrn.4105763`.

1DLT: Rapid Deployment of Secure and Efficient EVM-Based Blockchains

Simone Bottoni ✉
RTM, Lugano, Switzerland

Anwitaman Datta ✉
Nanyang Technological University, Singapore, Singapore

Federico Franzoni ✉
Unaffiliated Researcher, Barcelona, Spain

Emanuele Ragnoli ✉
RTM, Lugano, Switzerland

Roberto Ripamonti ✉
RTM, Lugano, Switzerland

Christian Rondanini ✉
RTM, Lugano, Switzerland

Gokhan Sagirlar ✉
RTM, Lugano, Switzerland

Alberto Trombetta ✉
Insubria University, Varese, Italy

Abstract

Limited scalability and transaction costs are some of the critical issues that hamper a wider adoption of distributed ledger technologies (DLTs). That is particularly true for the Ethereum [58] blockchain, which, so far, has been the ecosystem with the highest adoption rate. Several solutions have been attempted in the last few years, most of which adopt the approach to offload transactions from the blockchain mainnet, a.k.a. *Level 1* (L1), to a separate network. Such solutions are collectively known as *Level 2* (L2) systems. While improving scalability, the adoption of L2 introduces additional drawbacks: users have to trust that the L2 system has correctly performed transactions or, conversely, high computational power is required to prove transactions' correctness. In addition, significant technical knowledge is needed to set up and manage such an L2 system. To tackle such limitations, we propose 1DLT[1]: a novel system that enables rapid deployment of an Ethereum Virtual Machine based (EVM-based) blockchain that overcomes those drawbacks.

2012 ACM Subject Classification Computer systems organization → Distributed architectures

Keywords and phrases Blockchain, EVM, Layer Two, Scalability, Network Fees

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.3

Funding Anwitaman Datta's research is supported by Ministry of Education (MOE) Singapore's grant MOE-T2EP20120-0003 (award number #020931-00001).

1 Introduction

The current high demand for Ethereum [58] leads to slow transaction throughput (15-30 transactions per second [15]), expensive gas prices, and poor user experience for the majority of dapps (decentralised apps), Web3 projects, and end users. This limits the potential use cases, like in DeFi (decentralised finance), where high fees and scalability drawbacks enable only entities with vast economic power to trade profitably.

A notable example of extremely high gas prices and network congestion occurred with the launch of a new NFT for the *Bored Ape Yacht Club* metaverse [52]: during the launch, the Ethereum blockchain crashed due to traders outbidding each other by paying higher gas fees to execute their transactions faster. Users spent up to 7,000$ (2.6 ethers) to mint a 5,846$ NFT land deed for the virtual world, which sometimes resulted nevertheless in a failed transaction. A user trying to send 100$ in crypto between two wallets would need to pay a fee of 1,700$. As of July 2022, the cost of the above-mentioned NFT floats around 3,000$.

[1] Work done while all the authors were at QPQ AG.

Therefore, scaling solutions become crucial to increase network capacity in terms of speed and throughput. However, improvements to scalability should not be at the expense of decentralisation or trustlessness. Traditionally, scalability solutions are based on off-chain systems, collectively known as "Layer 2" (L2). L2 solutions are implemented separately from the "Layer 1" (L1) Ethereum mainnet, and do not require changes to its protocol. In L2 solutions, transactions are submitted to nodes of the L2 system instead of directly to L1 nodes. Thus, L2 solutions handle transactions outside the Ethereum mainnet and take advantage of its architectural features to ensure decentralisation and security. Existing L2 systems show a wide array of trade-offs among critical aspects like throughput, energy consumption, security guarantees, scalability, gas fees, and loss of trustlessness.

In this work, we present *One DLT* (1DLT), a novel, modular system for the rapid deployment of EVM-based blockchains, that avoids the pitfalls of many of the existing L2 solutions. Section 2 reviews the trade-offs of current solutions; Sections 3, 4, and 5 describe our system; Section 6 describes the Consensus-as-a-Service mechanism at the core of 1DLT; Section 7 show the 1DLT Bridge architecture; Section 8 exhibits a set of preliminary experimental results; finally, Section 9 concludes the work and describes our next steps.

2 Layer 2 limitations

There are several solutions available in the L2 ecosystem [13] (e.g., Optimistic Rollups [33], ZK-rollups [51], State channels [43], Sidechains [39]), with many different advantages and limitations. Due to space limitations, we do not present the main solutions adopted and we refer to the comprehensive surveys [53] and [54]. In the following, we list the most fundamental limitations and correlate them to some of the solutions adopted by the Ethereum ecosystem:

- **Limited expressive power**: some solutions do not support EVMs (e.g., several ZK-rollups, Plasma [34], Validium [48]); others support application-specific computations and require specialised languages (e.g., StarkWare's Cairo [55]);
- **Reduced trustlessness**: some solutions use operators and validators that can influence transaction ordering, leading to potential abuses (e.g., Optimistic Rollups, Sidechains);
- **Liveness requirement**: some solutions need to periodically watch the network or delegate this task to someone else to ensure security (e.g., Plasma);
- **High computational power to compute proofs**: some solutions require high computational power to compute proofs, which can be too expensive for dapps with little on-chain activity (e.g., ZK-rollups, Validium);
- **Reduced decentralisation**: some solutions adopt centralised methods to mediate the implementation of weak security schemes (e.g., Sidechains);
- **Limited throughput**: some solutions claim to theoretically achieve high transactions per second (tps) but are practically limited in their implementations (e.g, StarkWare [56] theoretically achieves 2,000 tps, while in real-world deployments is limited to 650 tps);
- **Not L2**: some solutions cannot be technically considered as L2 since they use separate consensus mechanisms that are not secured by the respective L1 (e.g., Sidechains). As such, these solutions cannot inherit from the L1 its security guarantees (e.g., resilience against chain tampering for Ethereum);
- **Private channels**: some solutions implement private channels, which is not a viable solution for infrequent transactions (e.g., State channels);
- **Long on-chain wait times**: some solutions require long wait times for on-chain transactions due to potential fraud challenges (e.g., Optimistic Rollups, Validium);
- **Data availability**: some solutions generate proofs that require off-chain data to be always available (e.g., Validium).

While the list above is not exhaustive (indeed, the L2 landscape is so dynamic that novel solutions, prototypes, and products are introduced to the market frequently), it is indicative of how, while there clearly exist attempts at overcoming these limitations, there is no single solution that can fix them all. It is important to note that a consequence of some of the limitations is the generation of security risks. Indeed, chains with relatively small ecosystems that provide consensus can lead to fallacies of abuse and fraud. Attackers, or the node maintainers themselves, may tamper with blockchain data ordering or validation, which allows them to redirect funds, perform flash loans or double-spending attacks, etc.

Lastly, in most solutions, users willing to create a private or public Ethereum network must rely on Ethereum *clients* (also known as *implementations*)[2] like Geth [20] and Erigon [8]. This approach requires the user to have significant technical knowledge and resources to maintain the nodes, with all the related costs and requirements of technical know-how.

3 An overview of 1DLT

This work has been inspired by the user experience of Cloud Service Providers (CSP) and Web-based applications, guided by the principles of DLTs. Indeed, in Cloud services, users are directed via graphical interfaces and dashboards through all processes (setup, configuration, billing, management, etc.). Such interaction is performed without any need of deep knowledge of the underlying technologies. Similarly, our goal is to provide a system that:

- streamlines and simplifies the deployment of a (public/private) EVM-based blockchain, as customarily happens for web-based services and CSP dashboards, without discarding the programmability of the EVM;
- maintains security while improving scalability and lowering fees;
- removes the risks associated with the L2 governance and fraud or abuse detection.

This is achieved with a modular, multilayered, cloud-native architecture (see Section 4) that decouples the transaction layer from the consensus layer. Thus:

- 1DLT connects to different blockchains and leverages their consensus mechanisms. We refer to this as *Consensus-as-a-Service* (CaaS) (see Section 6 for further details);
- 1DLT removes the risks associated with the L2 governance and fraud detection. Indeed, all transactions processed by 1DLT are sent to a L1 public blockchain, which is used as a consensus resource, allowing to inherit its security guarantees;
- 1DLT does not suffer from long wait times used in L2 to detect and avoid frauds. Fraud detection can be performed by checking receipts and confirmation messages of the public blockchain used as the consensus provider and local transactions' meta-data sent by CaaS;
- 1DLT is EVM-based, supports smart contracts written in the Ethereum programming language, Solidity, without requiring the adoption of L2 specific languages, such as Cairo;
- 1DLT transaction throughput is limited by that of the public blockchain that it connects to via CaaS. Thus, 1DLT outperforms Ethereum by connecting to blockchains with higher throughput and better scalability. Its performance can be further improved by connecting to different blockchains over time based on their load. The modular architecture makes it ready to plug in new, faster blockchain networks as they come into being;
- 1DLT allows to significantly reduce the transaction fees required to perform operations like payments, smart contract deployments, and token swaps, thanks to CaaS.

In the following, we show an example of user experience with 1DLT.

[2] A client is an implementation of Ethereum that verifies all transactions in each block, keeping the network secure and the data accurate [11].

▶ **Example 3.1.** Due to high transaction fees and long confirmation times, Alice wishes to move her dapp performing *NFTs Auctions* from the Ethereum mainnet to 1DLT to reduce operating costs. However, she does not want to change her codebase. To do that, Alice deploys a small, private EVM-based blockchain with 1DLT.

(i) Alice registers with the authentication system[3] and receives a redemption code for 1DLT blockchain creation;
(ii) Alice specifies the blockchain name and description, and token name and symbol. Then, with the redemption code, she creates a 1DLT Ethereum node in the private blockchain, configuring parameters such as cloud provider, virtual machine, etc.;
(iii) after specifying blockchain and nodes' parameters, Alice waits a short period for the execution of the setup procedure to start the deployment of her dapp;
(iv) Alice is now ready to deploy her dapp using the same procedure she used in Ethereum, that is, by sending a deployment transaction through the Web3 API;
(v) finally, upon receiving the deployment confirmation within few seconds, she and her customers are ready to interact with the dapp.

4 Architecture of 1DLT

1DLT has a modular architecture, consisting of two main components: a private EVM-based node (called *1DLT Ethereum Node*) and Consensus-as-a-Service (CaaS), a module that connects to external DLTs. When no confusion arises, from now on we refer to the 1DLT Ethereum Node simply as *Ethereum Node*. CaaS allows the Ethereum Node to leverage an external DLT to achieve consensus on transactions. Therefore, the 1DLT architecture abstracts and decouples the transaction layer from the consensus layer.

DLT adopts a very simple trust model in which the CaaS module is trusted. Hence, there are no checks from the Ethereum Node about the correctness of the responses from the CaaS module. Such trust assumption can be overcome by adding a verification mechanism to the Ethereum Node. In this work we focus on showing that 1DLT provides a low-cost scaling approach allowing high transaction outputs and fast transaction finality.

An overview of the 1DLT components and their interactions is shown in Figure 1. The structure of the Ethereum Node is detailed in the next section.

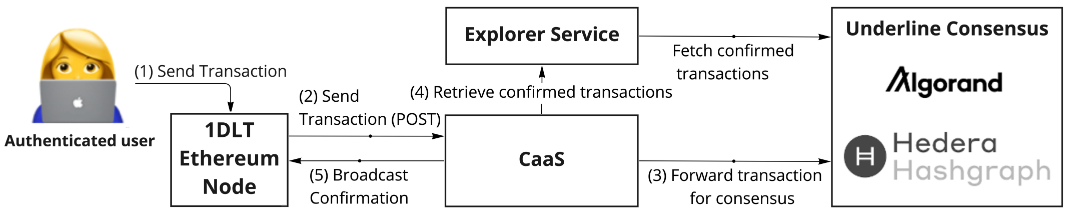

Authenticated user
(1) Send Transaction
1DLT Ethereum Node
(2) Send Transaction (POST)
(5) Broadcast Confirmation
Explorer Service
(4) Retrieve confirmed transactions
CaaS
Fetch confirmed transactions
(3) Forward transaction for consensus
Underline Consensus
Algorand
Hedera Hashgraph

Figure 1 1DLT architecture.

▶ **Example 4.1.** We continue with the scenario introduced in Example 3.1, where Alice sets up a node, and interacts with it by deploying a smart contract and sending a transaction. We expand on the operation flow sketched in Figure 1:

[3] A user must be authenticated to perform any action, thus a trusted authentication system is entitled to handle the user registration and management. Note that this trusted entity, while serving as a gateway for participants, is not relied upon for accountability of the actions of the participants. With our approach, the latter is achieved in a trustless manner using CaaS.

(i) Alice calls the Web3 API provided by the Ethereum Node to send a transaction, i.e. *eth_sendRawTransaction*;
(ii) the Ethereum Node receives the transaction and forwards it to CaaS using a POST message. This allows to achieve consensus on the transaction. The mechanisms used by CaaS to select a DLT for transaction dispatching is shown in Section 6;
(iii) CaaS forwards the transaction to the chosen DLT for confirmation;
(iv) when the transaction is confirmed, CaaS retrieves it using a blockchain explorer service (e.g., Hedera mirror service [24]);
(v) finally, CaaS sends the confirmation of the transaction to the Ethereum Node, which updates its state accordingly.

5 1DLT Ethereum Node

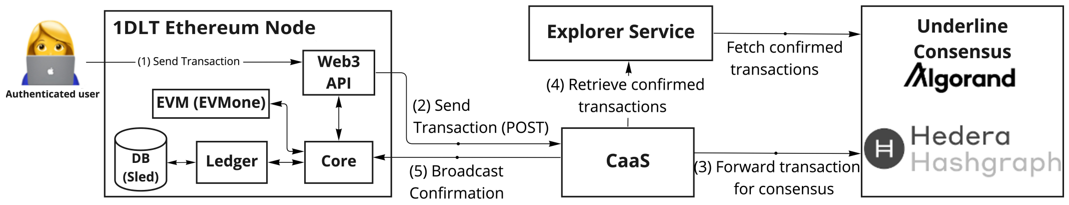


Figure 2 1DLT Ethereum node architecture.

While there are well-known and widely used implementations of Ethereum nodes, like Geth and Erigon, to overcome some of the limitations of Section 2, we engineered our own Ethereum node with a simpler architecture (see Figure 2). Section 5.2 describes the differences with a standard implementation. Usually, an Ethereum node contains a Web3 API, a state handling mechanism (i.e., the set of tries storing information on the state [29]), a database, an Ethereum Virtual Machine (EVM) [18], a p2p network, a transaction pool, and a consensus protocol. In the following, we describe the modules of our proposed architecture:

- **EVM module**: it is a sandboxed virtual stack machine that computes the system state transitions by executing an instruction specified in a transaction. In order to connect the node to a local, private EVM, the Ethereum Client-VM Connector API (EVMC) is used, as shown in Figure 3. The EVMC is the low-level interface between Ethereum Virtual Machines (EVMs) and Ethereum clients, which – on the EVM side – supports classic EVM1[4] and ewasm[5]. On the client side, EVMC defines the interface for accessing the Ethereum environment and state. A very relevant feature of EVMC is that nodes can connect with other non-Solidity based virtual machines.
  1DLT deploys a standalone C++ EVM implementation, called *EVMone* [19]. The EVMone EVM can be imported as a module by an Ethereum client and provides efficient execution of smart contracts written in an EVM-compliant language.
- **Web3 API module**: It handles incoming transactions and the communication with CaaS. It mostly works in the same way as in an Ethereum implementation, except for a customization that allows it to interact with CaaS. The module exposes a Web3-compatible API supporting modern Ethereum development tools and wallets (e.g., Metamask [30],

[4] Ethereum 1.x is a codename for a comprehensive set of upgrades to the Ethereum mainnet intended for near-term adoption.
[5] Ethereum flavoured WebAssembly is a subset of the WebAssembly format used for contracts in Ethereum.

Hardhat [22] and Web3.js [49]). E.g., the *eth_sendRawTransaction* API method is supported as in other implementations, but mining-related methods like *eth_isMining* are not supported, since consensus is handled through CaaS without any need for mining.

- **Ledger and state transitions module**: To store transactions and state, we use an Ethereum-compatible ledger implementation based on Merkle Patricia Trees (also known as (Merkle) Tries). This allows us to rely on the same structures as a standard Ethereum node (i.e., a State Trie, Receipt Trie, Transaction Trie, and Storage Trie [58]).
  Instead of the LevelDB [28] used in other Ethereum implementations, we opted for Sled [40], an embedded key-value store written in Rust optimised for modern hardware. It uses lock-free data structures to improve scalability and organises storage on disk in a log-structured way optimised for SSDs. We do not perform complex operations to achieve state transitions, such as the staged sync [9] in Erigon, as well as for block cutting. Since, CaaS validates transactions using an external consensus resource, it is possible to perform the block cutting in multiple ways. We opted to cut the block every Δ seconds (e.g., 10 seconds), after checking that the block is not empty.
- **Core module**: This module manages and coordinates the interaction of the other modules. It retrieves consensus updates of the processed transactions from CaaS (see Section 6) to perform state changes.

5.1 The execution flow

In what follows, we briefly describe the steps needed to perform a state update in a 1DLT Ethereum node upon receiving a transaction (see the sequence diagram in Figure 4).

1. the transaction is sent to the Ethereum node through the Web3 API;
2. the Web3 API module handles the transaction and sends a POST request to CaaS with the transaction wrapped inside the data field;
3. CaaS handles the transaction and connects to one of the chosen DLTs (e.g., Hedera) to confirm the transaction;
4. CaaS communicates with an external service (e.g. a Hedera mirror node) to retrieve the transaction confirmation;
5. CaaS then sends the confirmation message (i.e., the hash of the transaction with the consensus proof) of the transaction only to the Ethereum nodes that are part of the source network of the message.
6. the EVM of the Ethereum node executes the transaction, updating the state;
7. a block is then created if Δ seconds are passed (the time interval is a configurable parameter whose default is set to 10 seconds);
8. finally, the state transition result is stored in the Sled database.

5.2 Differences with current Ethereum implementations

Several relevant changes differentiate our node implementation from the standard one:

- **External Consensus**: in general, a consensus protocol has to be included in the internal architecture of a standard Ethereum node – like PoS for a mainnet, or PoA for a testnet node. Instead, we do not rely on an internal module, and delegate consensus to CaaS. This enables us to have the same liveness and safety guarantees of the chosen DLT while decreasing the complexity of the node.
- **No transaction pools**: in general, in a standard Ethereum node the transactions waiting for confirmation are placed into a transaction pool. Since our solution relies on an external consensus, all newly arrived transactions are forwarded directly to CaaS for confirmation without putting them in a queue. The benefits from this choice are a significantly simplified design and overall increased performance, as shown in Section 8.

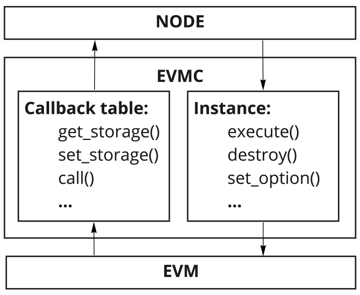


Figure 3 EVMC API.

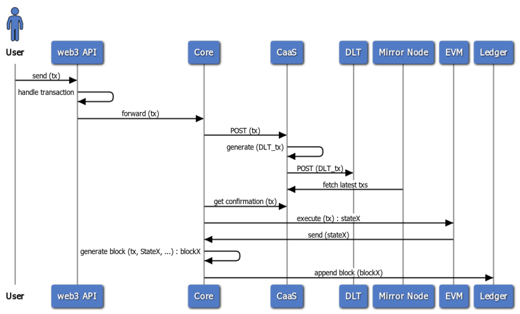


Figure 4 Sequence Diagram of a state update in a 1DLT Ethereum node.

- **Lightweight Core module**: as already mentioned in Section 5, the Core module is thoroughly simplified, since there is no transaction pool to manage, and it does not have complex state transitions (e.g., staged sync) as the consensus is retrieved from CaaS.
- **Difference in the Web3 support**: several methods are not supported by our implementation, such as those related to mining (e.g., *eth_getMining*, *eth_coinbase*), to uncles (e.g., *eth_submitHashrate*), and to Ethereum protocol (e.g., *eth_protocolVersion*).

6 Consensus-as-a-Service

Consensus-as-a-Service (CaaS) is the module that allows a 1DLT Ethereum node to access an external, public consensus protocol with an on-demand approach. Its key feature is the introduction of an abstract layer that enables the access to different DLTs through a single, uniform interface. This layer allows 1DLT Ethereum node to offload consensus complexity allowing it to achieve higher throughput and faster transaction finality with low transaction costs. Moreover, relying on an external consensus provider enables 1DLT to inherit the security model of the chosen DLT. Lastly, we remark that the CaaS approach eliminates any issue that may arise from the presence of a trusted third party, since transactions are public and easily auditable.

While in principle CaaS could attempt to tamper with handled transactions, such an attempt would make the transaction proof invalid, preventing the transaction execution, as each transaction proof required for an audit process can be retrieved from the target DLT. We acknowledge that this does not prevent CaaS to act in a malicious way and we leave as a future work the addition of a more efficient verification mechanism.

After choosing a suitable DLT, CaaS interacts with it by creating a channel that is used to publish messages containing transactions' information using the CaaS's DLT interface. The exchanged messages are stored in a time-series database (the current implementation uses timescale v2.6.0-pg1) to guarantee benefits over traditional relational database management systems like time-oriented features, higher data ingest rates and query performance. Additionally, each delivered message comes with the receipt of the transaction from the chosen DLT (e.g., an Hedera transaction receipt), which is an auditable proof that the transaction has been correctly processed by the DLT.

CaaS manages communication channels, I/O operations, and DB operations in a concurrent way, by spawning and managing multiple threads dedicated to the message exchange of different 1DLT networks. In particular, each 1DLT network has at least one dedicated communication channel, allowing high message processing throughput with low latency while also isolating multiple co-existing 1DLT networks from each other.

7 Bridge

Blockchains are siloed environments that cannot communicate with each other, as each network has its own protocols, native assets, data, and consensus mechanisms. Blockchain bridges, or cross-chain bridges [26][2], are a possible solution for enabling interoperability between different blockchains. The interoperability trilemma [45] allows for different bridge designs, for which, a non-standard classification can be based on [3]:

- **Trust model – How they work**: the type of authority used to synchronise the operations. The bridge is referred to as a "trusted bridge" if there is a central-trusted authority (e.g., Binance bridge [1]). If not, smart contracts make the bridge a "trustless bridge" by removing the necessity for a reliable third party (e.g., Connext [5], Hop [25], and other bridges with an atomic swap mechanism).
- **Validation – Validator or oracle-based bridges**: the type of mechanism the bridge relies on to validate cross-chain transfers, such as external validator or oracles.
- **Level – What they connect to**: the type of systems it connects to, such as a connection between blockchains or between a blockchain and an $L2$ system.
- **Sync – How they move assets**: the type of mechanism used to transfer assets between blockchains, such as *Lock and mint*, *Burn and mint*, or *Atomic swaps*.
- **Functionality – Their function**: the specialized interoperability task they are meant for, such as Chain-To-Chain, Multi-Chain, Specialized, Wrapped Asset, Data Specific, dapps Specific, and Sidechain.

1DLT offers a unique solution for bridges, enabling users to deploy a bridge by providing them with all the necessary tools and components (e.g., smart contracts). Since it operates via smart contracts, which serve as trusted parties, the 1DLT bridge belongs to the set of trustless bridges. It allows for the bi-directional transfer of ERC20 and ERC721 tokens between 1DLT nodes and EVM-compatible blockchains. 1DLT uses a *Lock and mint* mechanism: on the origin chain (e.g., Ethereum), a lock over the asset is performed, while on the destination chain (e.g., 1DLT), a mint is performed. Figure 5 provides a high-level breakdown of the bridge's core elements and how they interact. Essentially, the bridge is composed of a two of smart contracts, *Bridge.sol* and *Token.sol*, that are deployed on both the source and destination blockchain. Their interaction is coordinated via a cross-chain message dispatcher, called *Mediator*, using HTTP and WebSocket. The *Bridge* smart contracts implementation differ, as on the destination chain (i.e., *Bridge1DLT* for 1DLT) the contract design is for burn and mint, while on the origin chain is lock and withdraw (i.e., *BridgeETH*) for Ethereum). For the *Token* smart contract the implementation is the same for both the chains.

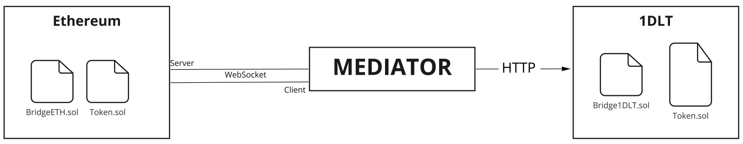


Figure 5 1DLT bridge architecture.

In what follows, we briefly describe the steps needed to perform the deposit of some ERC20 tokens from Ethereum to 1DLT (see Figure 6). The process relies on locking the asset on the source blockchain, and then mint the corresponding amount in the destination blockchain[6].

[6] To prevent the user from minting an arbitrary amount of token, only the bridge smart contract is entitled to call the mint method in the token smart contract.

1. The user sends a transaction to Ethereum, calling the *lock* method defined in the *BridgeEth* smart contract;
2. The transaction locks the tokens on Ethereum, transferring them to the *BridgeEth* address;
3. The *BridgeEth* emits a custom *Deposit* event with the address of the receiver on 1DLT and the amount;
4. The *Mediator* detects the event and retrieves the information;
5. The *Mediator* builds a transaction to call the *mint* method defined in *Bridge1DLT* with the event information as parameters;
6. The *Mediator* sends the new transaction to 1DLT;
7. 1DLT executes the transaction, which calls the *mint* method of *Bridge1DLT*;
8. The method calls the *mint* defined in *Token*;

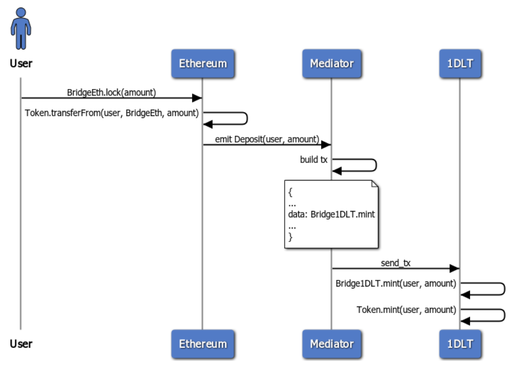


Figure 6 Deposit of ERC20 tokens from Ethereum to 1DLT.

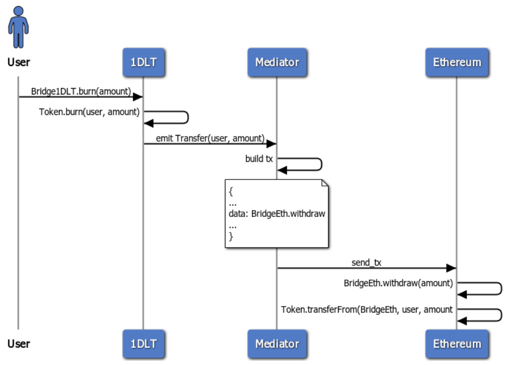


Figure 7 Withdraw of ERC20 tokens from 1DLT to Ethereum.

In what follows, we briefly describe the steps needed to perform the withdrawal of some ERC20 tokens from 1DLT to Ethereum (see Figure 7).

1. The user sends a transaction to 1DLT calling the *burn* method defined in *Bridge1DLT*;
2. The transaction burns the tokens on 1DLT;
3. The *Bridge1DLT* emits a custom event with the address of the receiver and the amount;
4. The *Mediator* detects the event and retrieves the information;
5. The *Mediator* builds a transaction to call the *withdraw* method defined in *BridgeEth* with the event information as parameters;
6. The *Mediator* sends the transaction to Ethereum;
7. Ethereum executes the transaction, which calls the *withdraw* method of *BridgeEth*;
8. The method calls the *transferFrom* defined in *Token*;

As virtually all complex, interacting systems, bridges are exposed to security risks related to:

- **Smart Contracts**: bugs in their code can be exploited for malicious behaviours;
- **Underlying Blockchain**: the underlying blockchain may be breached or act improperly;
- **Users**: users not following best practices can incur non-secure behaviours;
- **Censorship and Custodial**: bridge operators may act in malicious ways (e.g. they can suspend their activities or collude to gain sensitive information about the bridge's users)[7].

There are numerous examples of bridge attacks that resulted in multimillion dollar losses. [27]. It is worth to mention that bridge hacks mainly happen due to a vulnerability identified and exploited within the bridge contract, such as in the Wormhole attack [50] or the Optimism

[7] Applies to bridges that require the presence of trusted operators.

smart contract bug [32]. In the remaining cases, user mistakes take place, such as in the Optimism Wintermute case [31], where a Wintermute user inserted the wrong destination address for a transaction.

To overcome some of the previously mentioned risks, we give the user the ability to set up its 1DLT bridge without 1DLT system taking control of the bridge or custody of the assets. 1DLT offers reliable smart contracts that adhere to community-tested security best practices, such as Optimism [46] or Polygon[35]). Thus, an internal and external auditing procedure of the smart contracts, bridge, and the node is conducted to ensure the users' safety. For the external audit process, we use well-known auditors, such as CertiK[4], Hacken[21], and Trail of Bits [47]. As part of the internal audit process, we use a smart contract bytecode verification similar to the one of Etherscan [17] and Sourcify [42]. To this end, to have the smart contracts verified, a user must share with 1DLT the transaction hash of the exploited smart contracts via a dedicated page.

8 Experiments and Performance Discussion

A set of preliminary experiments were performed to benchmark and test 1DLT. We run experiments according to the following metrics:

1. **Total transaction cost**: the cost of sending a transaction, which is computed as: $Cost_{total} = cost_{transaction} + fee_{gas} + fee_{DLT}$ where: $cost_{transaction}$ is the cost associated to the transaction, which may involve the execution cost of a smart contract or an amount of tokens; fee_{gas} and fee_{DLT} are the fee costs associated to the transaction from the node and for the target DLT. We note that the data field is where the difference between a transaction and an interaction lies; in a transaction, the field is empty; in an interaction, it has a value.

2. **Transaction finality**: the amount of time a user has to wait, on average, to obtain a confirmation of a transaction, measured in seconds. In the case of 1DLT, the finality time includes the transaction processing time and consensus finality time of the public DLT.

3. **Total smart contract deployment cost**: the cost of deploying a smart contract, which is computed as: $Cost_{total} = cost_{transaction} + fee_{gas} + fee_{DLT} + cost_{createContract} + cost_{data}$ where: $cost_{transaction}$ is the cost associated with the contract creation transaction; fee_{gas} and fee_{DLT} are the fee costs associated to the transaction from the node and for the target DLT; $cost_{createContract}$ is the cost associated to a contract creation, fixed to 32000 gas; and $cost_{data}$ is the cost associated to the contract complexity. As of today, we do not support only the legacy format (before EIP 2930 [7]).

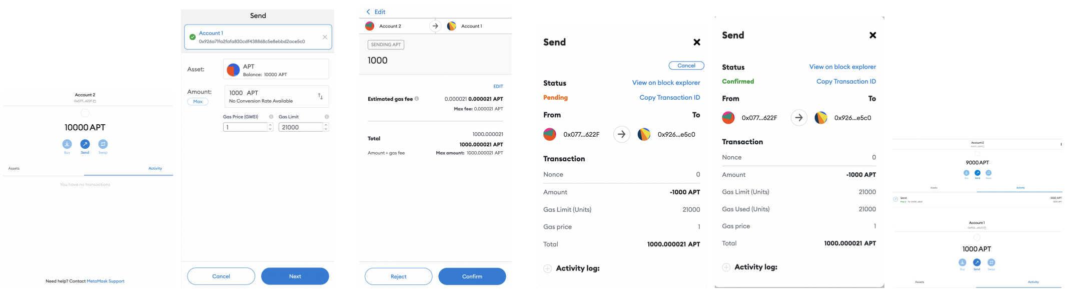
4. **Throughput**: the transaction rate of a blockchain, measured in transactions per second (TPS). It is known that throughput is not the inverse of latency. For example, the transaction throughput for Bitcoin is about $7tx/second$ [37] due to relatively small blocks and long block time. Instead, Ethereum has a short block time but tiny blocks, which results in a $15tx/second$ [38]. 1DLT's throughput is limited by the total throughput of public blockchains that CaaS connects to. In fact, all transactions submitted to CaaS by the 1DLT Ethereum nodes are forwarded to the public blockchains like Hedera and Algorand. Therefore, 1DLT throughput varies proportionally with the throughput of the blockchain CaaS connects to.

The experiments are executed on an Azure Virtual Machine[8] configured as *Standard_ D2_v3*[9], with 2 vCPUs, 8 GB of RAM, 256 GB SSD, and running *Ubuntu 21.10*. We use Hedera Consensus Service (HCS) [23] on the Testnet as the consensus resource. We simulate the Web3 API interaction using Web3.js API [49]. We use Metamask [30] as the wallet application to verify the state of the transactions and Hardhat as development environment [22], as it is the de-facto standard tool for developing dapps [41].

8.1 Total transaction cost

We consider the token in Alice's 1DLT network, with token name and symbol *Alice_Token* and APT, respectively. In Figure 8, we show the steps done from Metamask's user interface to transfer tokens from Alice to Bob: first, we specify Bob's account as the destination, the APT token as the asset, and 1,000 as the amount. Next, we check the calculated fees and send the transaction. Initially, the transaction state is on pending, then, after 6 seconds, the state changes from pending to confirmed, allowing the balance update for Alice and Bob.

The total cost ($Cost_{total}$) for Alice is as follows: fee_{Gas} is 0.000021 APT, $cost_{transaction}$ is 1000 APT, and fee_{DLT} is 0.00051779 *HBAR*, which is 0.00000003 APT (assuming that 1 ETH = 1 APT). So the cumulative cost is $1,000.00002103$ APT and the cumulative cost for the fees is 0.00002103 (0.056 USD). On the Ethereum Testnet (Ropsten [14]) the cumulative cost is 0.00005093 (total of 0.14 USD), while on the Ethereum mainnet, with a gas fee of 47 *Gwei*, it's 0.000819 (for a total of 2.95 USD).


Account 2
0x077...622F
10000 APT
You have no transactions
Need help? Contact MetaMask Support

Send
Account 1
0x926a71fa2fafa830cdf438868c5e8ebbd2ace5c0
Asset: APT Balance: 10000 APT
Amount: 1000 APT No Conversion Rate Available
Gas Price (GWEI) 1
Gas Limit 21000

Account 2 → Account 1
SENDING APT
1000
Estimated gas fee 0.000021 0.000021 APT
Max fee: 0.000021 APT
Total 1000.000021
1000.000021 APT
Amount + gas fee Max amount: 1000.000021 APT

Send
Status: Pending
From 0x077...622F To 0x926...e5c0
Transaction
Nonce 0
Amount -1000 APT
Gas Limit (Units) 21000
Gas price 1
Total 1000.000021 APT
Activity log:

Send
Status: Confirmed
From 0x077...622F To 0x926...e5c0
Transaction
Nonce 0
Amount -1000 APT
Gas Limit (Units) 21000
Gas Used (Units) 21000
Gas price 1
Total 1000.000021 APT
Activity log:

Account 2
9000 APT
Send -1000 APT
Account 1
1000 APT

Figure 8 Setup and send transaction from Alice to Bob state transition and account update.

8.2 Transaction finality

We evaluate the consensus finality of 1DLT using a client app that generates payment transactions, and submits them to a 1DLT Ethereum node in its 1DLT network. We compute the overall time from the generation of the transaction to the balance update in the Metamask wallet as: $Overall\_time = Generate\_Send_{tx} + 1DLT\_Finality_{tx} + Update\_wallet_{tx}$ where:

- $Generate\_Send_{tx}$ is the time our client application takes to generate and send the payment transaction to a 1DLT Ethereum Node of 1DLT;
- $1DLT\_Finality_{tx}$ is the time for 1DLT to process a transaction;
- $Update\_wallet_{tx}$ is the time the Metamask wallet takes to update the balance via a call sent by 1DLT.

[8] https://azure.microsoft.com/
[9] https://docs.microsoft.com/en-gb/azure/virtual-machine/dv3-dsv3-series

In Figure 9, we present the experiment results, showing the *Overall_Finality_Time* that we measured executing 200 transactions. We observe that average execution time for *Overall_Finality_Time* is *4.526* seconds.

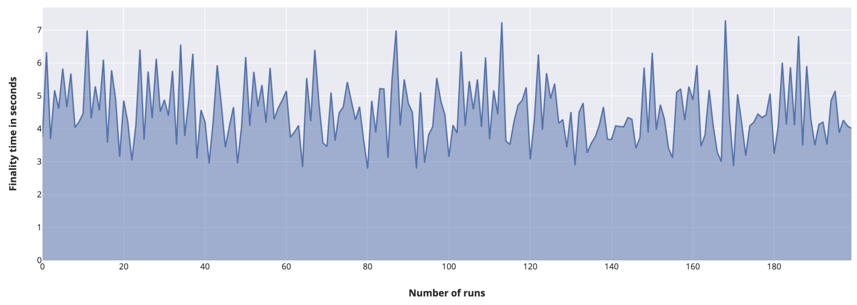


Figure 9 Transaction Finality experiment Diagram.

8.3 Smart contract deployment cost

We consider the example in 3.1 where Alice deploys the smart contract for her NFTs Auction dapp. To deploy the auction smart contract, we write a deployment script in JavaScript. We add the 1DLT node information, Node IP address, chain ID and private key of Alice account to the Hardhat configuration file, *hardhat.config.js*. Then, from terminal, we execute the deployment script with the command:

```
$ pnpm hardhat run --network AliceNetwork deploySmartContract.js
```

Once the deployment is completed, we receive the address of the created smart contract, like:

```
$ Contract deployed to address: 0x6cd7d44516a20882cEa2DE9f205bF401c0d23570
```

The transaction cost for deploying the smart contract on 1DLT is 0.000013402 APT (suppose that 1 ETH = 1 APT). On the Ethereum Testnet (Ropsten) the cumulative cost is 0.0015402 (for a total of 1.74 USD), while on the Ethereum mainnet, with a gas fee of 30 *Gwei*, it's 0.0117055 (for a total of 13.91 USD). Note that the cost to interact with a smart contract, that is, to call a method that changes the state (e.g., a set method), is calculated the same as a transaction since a setter method is implemented by sending a transaction.

8.4 Transaction per second (TPS)

We evaluate the performance of CaaS using Hedera as the consensus resource. We use a different Azure virtual machine configuration than before. We run CaaS on an Azure VM in the Switzerland North region configured as Standard DS3 v2, with 4 vCPUs, 14 GB of RAM, 1 TB SSD, and running Ubuntu 20.04. Then, to simulate the client, we use a VM configured with Standard D2s v3, 2 vCPUs, 8 GB memory, and running Ubuntu 20.04. In this experiment, we configured CaaS to run on a small VM to demonstrate that it is very lightweight and can achieve high performance even with this setup. Ideally, and in the production environment, CaaS will run on a Kubernetes cluster that allows to scale up with the increased transaction requests. The client runs a Python v3.8.10 script that generates a total of ten thousand transactions across five Hedera topics (which corresponds to five communication channels in CaaS), sends transactions to CaaS, and waits for confirmations from CaaS. Running the experiment 10 times with a single client results in an average of 1120 tps. The results are promising, as this experiment proves that a single client can process around 1120 transactions per second with a minimal CaaS setup as discussed above.

8.5 Discussion

The energy consumption of operating the Ethereum network has decreased by 99.9% after the Merge, occurred in mid September 2022 [44].

However, the gas fees' costs have not changed and this results in high costs for deploying Ethereum as a computing platform. In fact, each finalized block includes 30 million Gas, which is the amount of Gas used for all the transactions in a block [10]. The current transaction fees for 30 million of consumed gas is more than 1 Ether, which (as of July 2022) is valued at 1,479 USD. This implies that the computation costs of the Ethereum Network are around 133 USD per second, which is ∼25 times more than 15 days of an EC2 instance (currently around 20 USD).

In order to estimate the cost of the Ethereum network itself (measured in gas), following [57], we consider a very basic computational task like adding two 256-bit integers. Since this operation costs 3 Gas [12] and the Ethereum network's total compute is 2 million gas/second, the Ethereum network may perform 600,000 additions per second. In contrast, Raspberry Pi 4 [36], a 45 USD single-board computer with four processors running at 1.5 GHz, can perform around 3,000,000,000 additions per second. As such, the Ethereum network, considered as a general-purpose computational environment, has roughly 1/5,000 of the computing power of a Raspberry Pi 4. At the current gas price, this means that performing 256-bit additions on the Ethereum network, costs about 60 USD per month.

Ethereum as a computational environment has the drawback to be expensive, in terms of gas fees. Since 1DLT follows a modular approach and separates the EVM-based computational layer from the consensus layer, it minimises energy consumption and computational effort. Thanks to the deployment of CaaS, the consensus engine does not require a mining algorithm in the consensus retrieval. Additionally, 1DLT offers the same level of computational power as Ethereum at a significantly lower cost, as shown in experiments 1 and 3 in Section 8. As an example, the cost to execute 50,000 transactions is 0.04 USD.

9 Conclusion and Future work

Scaling solutions are crucial to increase the Ethereum network's capacity in terms of speed and throughput, but they come at the cost of reduced decentralisation, increased transaction finality times. 1DLT, inspired by the user experience of Cloud Service Providers (CSP) and Web-based applications, overcomes the limitations of the existing scaling solutions enabling low gas fees, high transaction throughput, and fast transaction finality. Additionally, 1DLT removes the risk associated with the L2 governance and fraud detection, since all the transactions processed in 1DLT networks are submitted to the consensus protocols of L1 public DLTs. Lastly, the programmability and user experience of the Ethereum ecosystem is maintained thanks to an EVM-based architecture.

We demonstrated the feasibility of our architecture with a set of preliminary experiments in Section 8, benchmarking transaction costs and finality.

Future work includes: (i) a proper experimental benchmark to execute advanced experiments that fully accounts for the real-world landscape of L2 solutions and 1DLT; (ii) Extend the 1DLT Ethereum Node with a verification mechanism for proving the correctness of Caas module's responses; (iii) enhance the bridge with support for blockchains that are not compatible with EVM; (iv) the integration of 1DLT with Trusted Execution Environments; (v) full integration with a proprietary wallet that will enhance the Metamask solution; (vi) provision of user tools like Etherscan [16] for Ethereum or DragonGlass [6] for Hedera, to audit the status of the blockchains in the 1DLT ecosystem.

References

1 Binance bridge. https://www.bnbchain.org/en/bridge.
2 Blockchain bridges. https://ethereum.org/en/developers/docs/bridges.
3 Blockchain bridges classification. https://li.fi/knowledge-hub/bridge-classification.
4 CertiK. https://www.certik.com.
5 Connext bridge. https://bridge.connext.network.
6 DragonGlass. https://app.dragonglass.me.
7 EIPS 2930. https://eips.ethereum.org/EIPS/eip-2930.
8 Erigon. https://github.com/ledgerwatch/erigon.
9 Erigon stage sync. https://github.com/ledgerwatch/erigon/blob/devel/eth/stagedsync.
10 Ethereum gas. https://ethereum.org/en/developers/docs/gas.
11 Ethereum nodes. https://ethereum.org/en/developers/docs/nodes-and-clients.
12 Ethereum opcodes. https://ethereum.org/it/developers/docs/evm/opcodes.
13 Ethereum Scaling. https://ethereum.org/en/developers/docs/scaling.
14 Ethereum testnets. https://ethereum.org/en/developers/docs/networks.
15 Ethereum TPS. https://ethtps.info.
16 Etherscan. https://etherscan.io.
17 Etherscan smart contract verification. https://etherscan.io/verifyContract.
18 EVM. https://ethereum.org/en/developers/docs/evm.
19 EVMone. https://github.com/ethereum/evmone.
20 Geth. https://geth.ethereum.org/docs.
21 Hacken. https://hacken.io.
22 Hardhat. https://hardhat.org.
23 Hedera Consensus Service. hedera.com/consensus-service.
24 Hedera mirror service. https://hedera.com/learning/hedera-hashgraph/what-is-the-hedera-mirror-network.
25 Hop. https://app.hop.exchange.
26 Introduction to blockchain bridges. https://ethereum.org/en/bridges/.
27 Leaderboard of Ethereum bridge attacks. https://rekt.news/leaderboard.
28 Leveldb database. https://github.com/google/leveldb.
29 Merkle Patricia Trie. https://eth.wiki/fundamentals/patricia-tree.
30 Metamask. https://metamask.io.
31 Optimism Attack. https://cointelegraph.com/news/optimism-loses-20m-tokens-after-l1-and-l2-confusion-exploited.
32 Optimism Bounty. https://cryptoslate.com/critical-bug-in-ethereum-l2-optimism-2m-bounty-paid.
33 Optimistic Rollups. https://ethereum.org/en/developers/docs/scaling/optimistic-rollups.
34 Plasma. https://ethereum.org/en/developers/docs/scaling/plasma.
35 Polygon-Ethereum Bridge. https://docs.polygon.technology/docs/develop/ethereum-polygon/getting-started.
36 Raspberry. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.
37 Real time Bitcoin finality. https://statoshi.info/d/000000006.
38 Real time Ethereum finality. https://ethtps.info.
39 Sidechains. https://ethereum.org/en/developers/docs/scaling/sidechains.
40 Sled database. https://github.com/spacejam/sled.
41 Solidity report. https://blog.soliditylang.org/2022/02/07/solidity-developer-survey-2021-results.
42 Sourcify. https://docs.sourcify.dev/docs/intro.
43 State Channels. https://ethereum.org/en/developers/docs/scaling/state-channels.
44 The Ethereum merge. https://ethereum.org/en/upgrades/merge.

45 The interoperability trilemma. `https://blog.connext.network/the-interoperability-trilemma-657c2cf69f17`.

46 The Optimism bridge. `https://ethereum.org/en/developers/tutorials/optimism-std-bridge-annotated-code`.

47 Trail of Bits. `https://www.trailofbits.com`.

48 Validium. `https://ethereum.org/en/developers/docs/scaling/validium`.

49 web3.js API. `https://web3js.readthedocs.io/en/v1.7.4/`.

50 Wormhole hack. `https://rekt.news/wormhole-rekt`.

51 Zk Rollups. `https://ethereum.org/en/developers/docs/scaling/zk-rollups`.

52 Bored ape crush Ethereum. `https://www.cnet.com/personal-finance/crypto/bored-ape-yacht-club-just-broke-the-ethereum-blockchain`, 2022.

53 Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography and Data Security - 24th International Conference, FC*, Lecture Notes in Computer Science, pages 201–226. Springer, 2020.

54 Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. *IACR Cryptol. ePrint Arch.*, page 352, 2019.

55 Starkware. Starkware Cairo. `https://starkware.co/cairo`.

56 Starkware. Starkware Libs. `https://github.com/starkware-libs`.

57 Nicholas Weaver. The Web3 Fraud. `https://www.usenix.org/publications/loginonline/web3-fraud`, 2021.

58 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

Consistency of Automated Market Makers

Vincent Danos
CNRS, France
DI ENS, INRIA, PSL, Paris, France

Weijia Wang
ENS, Paris, France

Abstract

Decentralised Finance has popularised Automated Market Makers (AMMs), but surprisingly little research has been done on their consistency. Can a single attacker extract risk-free revenue from an AMM, regardless of price or other users' behaviour? In this paper, we investigate the consistency of a large class of AMMs, including the most widely used ones, and show that consistency holds.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design

Keywords and phrases Automated Market Makers, Decentralised Finance

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.4

Acknowledgements The first author wishes to thank Jérôme de Tichey for introducing him to the problem of consistency, Guillaume Terradot and Hamza El Khalloufi for numerous conversations.

1 Introduction

Blockchains offer in principle a neutral computational medium, where anyone can deploy and interact with smart contracts without interference from external parties. It has been long thought that the ability for parties to enter into interactions that have to follow rules captured un-ambiguously by code could change finance [8]. Indeed, there is now an emerging domain known as decentralized finance (DeFi) re-defining financial primitives and functionalities using smart contracts. Its progresses and potentials were recently recognised in a report of the IMF.[1] One key component of any financial system is the mechanism for matching participants willing to trade. DeFi has brought to the fore a novel class of protocols, namely automated market makers (AMMs), to perform this task. In this paper, we investigate the consistency of such AMMs.

An automated market maker (AMM) is a specific type of decentralized exchange, ie a market place which is fully automatised and implemented as a smart contract. An AMM protocol typically maintains reserves, also referred to as pools, of different assets and employs mathematical algorithms to determine the price of assets and identify which trades it is willing to execute with a particular trader. The AMM pools are populated by users known as liquidity providers (LPs). In return for providing liquidity, LPs receive LP tokens, also referred to as pool tokens, representing their ownership fraction of the pools, and receive accordingly a fraction of the fees paid by traders on each trade. Nothing prevents a user from taking on both roles, that is to say, to deposit/withdraw assets as an LP, while at the same time trading assets with the AMM.

[1] "By taking innovation to a new level [...] DeFi has had extraordinary growth in the past two years, potentially offering higher efficiency and investment opportunities." "DeFi offers broad access to players of any size and has no need for custodian service, potentially improving efficiency and financial inclusion." (IMF report, Apr 2022)

The top-performing AMMs in 2022 achieved weekly trading volumes in the billions of euros [3]. Despite those substantial volumes, little is known about the consistency of the underpinning mathematical mechanisms. Of course, providing liquidity can lead to losses depending on the evolution of prices. In the context of AMMs, loss due to price movement is called impermanent loss. However, in this paper, we address a different risk which is unrelated to price action, arbitrage (exploiting price discrepancies on different markets), or the exploitation of other players' moves (as in various types of front-running). We ask whether a single attacker with unbounded capital can initiate a sequence of interactions with a given AMM, which would lead to a risk-free and price-independent profit. If there is no such sequence, regardless of the initial state of the AMM, then we say the AMM is consistent. There are examples of such attacks, making the question of consistency a practically important one [4].

The latest and only known result on the consistency problem, published in 2020, showed that under reasonable conditions on the AMM mechanics, the AMM is consistent as long as the attacker is only allowed to trade (and not to provide liquidity) [2]. This weaker notion of consistency can be readily proved for a large class of AMMs. It leaves open the question of consistency where the attacker is allowed to combine LP actions and trading.

Outline

We begin with the definition of a large class of AMMs which we call price machines, and narrow down their definition to the case of a single attacker (§2). We turn to the definition of consistency and prove our main abstract result which gives a simple sufficient condition for the consistency of a price machine (§3). The third part of the paper (§4) is devoted to applications. We establish the consistency of DeFi's most popular AMMs: Uniswap [1], Balancer [7], and both versions of the Curve AMM [5, 6].

2 Basic definitions

Throughout the paper we use the terms assets and tokens as synonymous. We call market participants simply users. In this section, we define and discuss *price machines* which form the class of AMMs we investigate.

2.1 Preliminaries

The product ordering on $\mathbb{R}^n$, $n > 0$, is written $\preceq$. It is defined as usual as $v \preceq v'$ iff $v_i \leq v'_i$ for $1 \leq i \leq n$. The strict version is written $\prec$. A relevant intuition for the product ordering is that $v \preceq v'$ iff $p^T(v' - v) \geq 0$ for any "price" vector p with positive coordinates. The positive part of a vector $v \in \mathbb{R}^n$, written v_+, is defined as $(v_+)_i = \max(v_i, 0)$, for $1 \leq i \leq n$. We write $\mathbf{1}$ for the vector with all components equal to 1.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be non-decreasing (increasing) if it preserves the (strict) product ordering.

It is easy to see that a function $f : \mathbb{R}^n \to \mathbb{R}$ is increasing iff it is locally increasing, meaning it preserves $\prec$ on a neighbourhood of each point.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is called homogenous of degree $k \geq 0$ if for every $v \in \mathbb{R}^n$ and $s \neq 0$, $f(sv) = s^k f(v)$. It is called positively homogenous if the identity holds for $s > 0$.

We write $\mathbb{R}^*_+$ for the set of positive reals.

2.2 Trading functions

A *trading function* on $n \geq 1$ tokens is an increasing function $\psi : (\mathbb{R}_+^*)^n \to \mathbb{R}_+^*$. We say ψ is 1-homogenous if for $s > 0$, $\psi(sv) = s\psi(v)$.

For a vector of reserves $R \in (\mathbb{R}_+^*)^n$, $\psi(R) \in \mathbb{R}_+^*$ specifies the total amount of LP tokens currently distributed (up to a positive constant multiplicative factor). We also pick ϵ a (small) constant $0 \leq \epsilon \leq 1$ specifying the LP fees.

Given ψ, ϵ, we define a notion of AMM trading on n tokens.

Specifically, we distinguish two types of transitions: *Swaps* and *Transfers*. Both types take current reserves R to new reserves R'. Swaps correspond to trade events, transfers correspond to liquidity provision events (deposits and withdrawals).

Swaps: a swap $R \to^s R'$ must satisfy $\psi(R) = \psi(R' - \epsilon(R' - R)_+)$

The vector $(R' - R)_+$ is the amount of tokens received by the AMM, while the vector $(R - R')_+$ is the amount paid out to the trader. As ψ is increasing, it must be that $\psi(R) \leq \psi(R')$. The excess amount $\psi(R') - \psi(R)$ of LP tokens is divided between LPs in proportion to their current amounts of LP tokens.[2]

When $\epsilon = 0$ the swap constraint becomes simply $\psi(R) = \psi(R')$. This is the reason ψ is often referred to as the *invariant* of the AMM. For (small) $\epsilon > 0$, swaps induce a (small) increase of the invariant. In the limit case $\epsilon = 1$, $R' - \epsilon(R' - R)_+ = R \wedge R'$, hence the constraint can be rewritten $\psi(R) = \psi(R \wedge R')$, which implies $R = R \wedge R'$ because ψ in increasing, or equivalently $R' \succeq R$. In words, the AMM never pays out anything on a swap.

Transfers: a transfer $R \to^t R'$ must satisfy either $R \preceq R'$ (deposit), or $R \succeq R'$ (withdrawal).

Deposits (withdrawals) increase (decrease) the current value of the invariant. The excess amount $\psi(R') - \psi(R)$ of LP tokens (negative in the case of a withdrawal) is given to (taken from) the user who initiated the transfer.

A transfer is said to be *balanced* if there exists $\lambda \in \mathbb{R}_+$ such that $R' = \lambda R$, and *perfectly balanced* if $\psi(\lambda R) = \lambda\psi(R)$.

In the next section, we show (Prop. 4) that, if ψ is homogenous, imbalanced transfers can be decomposed as a swap without fees followed by a balanced transfer. Imbalanced transfers are best understood as a compound transition which is convenient to users.[3]

2.3 Example (verified on-chain [10])

Uniswap v2 is a two-token AMM with trading function $\psi(R) := (R_1 R_2)^{\frac{1}{2}}$, and fee $\epsilon = 0.003$. Say the tokens are named A and B. We start in the following configuration:

- The AMM's pool has $5A + 20B$ tokens.
- Alice has 10 LP tokens and is the only LP, since $\psi(5, 20) = 10$
- Bob has $50A + 200B$ tokens
- Charlie has $10A$ tokens

[2] In other words, no-one gets diluted. In reality AMMs do not literally offer LP tokens to their LPs at each swap. For reasons of efficiency, they keep track of the total number of LP tokens separately, to preserve the proportions of LP tokens for each user on swaps, and determine transfers appropriately.

[3] Indeed, in practice, imbalanced transfers are subject to a fee, In the words of the Balancer AMM documentation "Since Balancer allows for depositing and withdrawing liquidity to Balancer pools using only one of the tokens present in the pool, this could be used to do the equivalent of a swap: provide liquidity depositing token A, and immediately withdraw that liquidity in token B. Therefore a swap fee has to be charged, proportional to the tokens that would need to be swapped for an all-asset deposit." (See single asset deposit withdrawal.)

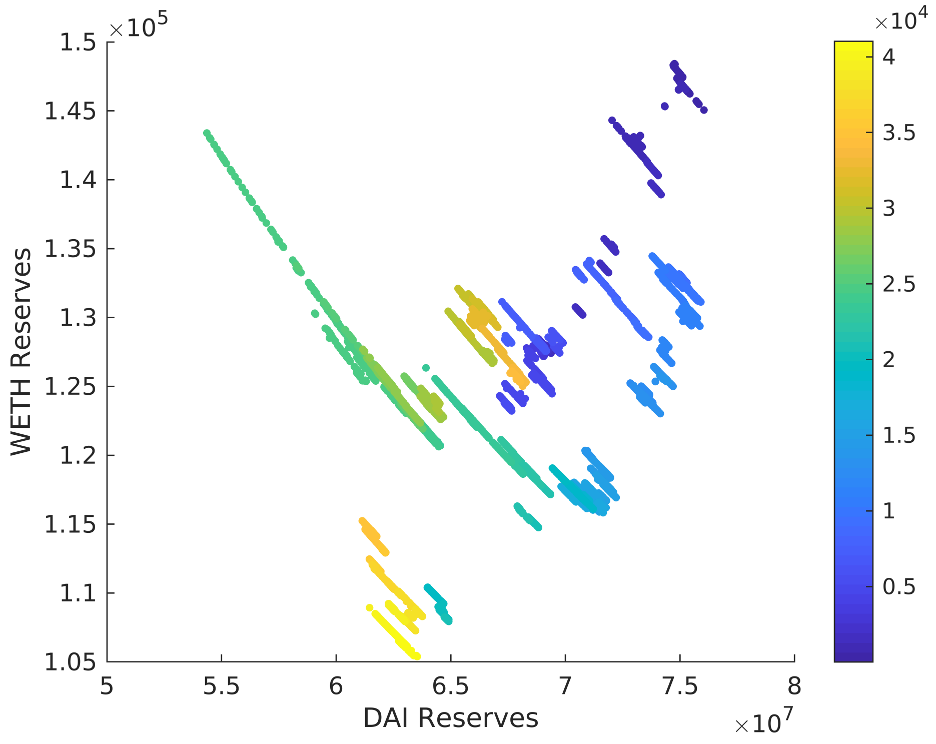


Figure 1 A sequence of aout 40,000 transitions (recorded during the first two weeks of November 2021) on the WETH/DAI Uniswap v2 Ethereum market; colours represent succession in time.

We want to illustrate the various types of transitions:

- (Transfer) Bob deposits $50A + 200B$ tokens (balanced deposit, $\lambda = 11$).
  The pool has now $55A + 220B$ tokens, and Bob receives $\sqrt{50 \cdot 200} = 100$ LP tokens.
- (Swap) Charlie swaps As for Bs, using $10A$ tokens.
  The pool has now $65A$ tokens and $\frac{55 \cdot 220}{65 - 10\epsilon} = 186.239803\,B$ tokens, while Bob receives $220 - 186.239803 = 33.760197\,B$ tokens.
  The excess amount of LP tokens generated by Charlie's swap is shared proportionally, so that Alice has now $\frac{10}{110}\sqrt{65 \cdot 186.239803} = 10.002308$ LP tokens, and Bob has now 100.023085 LP tokens.
- (Transfer) Alice withdraws all her 10.002308 LP tokens (balanced withdrawal)
  Alice receives $\frac{10.002308}{\sqrt{65 \cdot 186.239803}} = \frac{10}{110}$ from the pool, i.e. $\frac{10}{110} \cdot 65 = 5.909091\,A$ tokens and $\frac{10}{110} \cdot 186.239803 = 16.930891B$ tokens.
  The pool ends up with $59.090909\,A + 169.308912\,B$ tokens.

Fig. 1 gives a historical example (sampled during Nov 2021) of a far longer sequence (of circa 40,000 transitions) on a Uniswap v2 market (on the `ETH/DAI` token pair). Swaps correspond to motions along the hyperbolic contour lines of the trading function. Transfers correspond to jumps from one contour line to a lower (higher) one if a withdrawal (deposit).

2.4 Price machines

The simple example above shows that to properly record the effect of a sequence of transitions on an AMM one needs to incorporate in its state two additional components:

- the (proportions of) LP tokens held by each user,
- as well as their own reserves in the tokens of interest.

To keep things simple we now suppose $\epsilon = 0$. (Fees can be computed for each swap a posteriori anyways.) The resulting model is defined below as a labelled transition system, with its state space, labels, and labelled transitions.

▶ **Definition 1.** *A price machine with n-token trading function ψ and user set U is a labelled transition system with:*

- *state space $S = (\mathbb{R}_+^*)^n \times \mathbb{R}_+^U \times \mathbb{R}_+^{U\times n}$*
- *transition labels $\Sigma = \{s, t\} \times U \times \mathbb{R}_+^n$*
- *(labelled) transition relation described below*

States *are of the form $(R, \theta, \hat{R}) \in S$ where:*

- *$R \in (\mathbb{R}_+^*)^n$ is the vector of reserves of the price machine*
- *$\theta \in \mathbb{R}_+^U$ is the vector of fractions of LP tokens held by users, so that $\mathbf{1}^T\theta = 1$*
- *$\hat{R} \in \mathbb{R}_+^{U\times n}$ is the vector of users' net wealths*

The net wealth of user u is defined as the amount of tokens user u would own after *withdrawing all its LP tokens. The vector $R_u^{loc} := \hat{R}_u - \theta_u R$, represents u's local reserves, that is to say the amount of capital which u has not invested as an LP.*

Labels *are of the form $(s/t, u, \lambda)$ where:*

- *s/t indicates whether the transition is a swap (s) or a transfer (t)*
- *u is the user causing the transition*
- *$\lambda \in \mathbb{R}_+^n$ is a vector expressing multiplicatively the change in reserves induced by the transition*

That is to say, for a transition in reserves $R \to R'$, λ is the unique (positive) vector such that $R' = \lambda R$, where multiplication is understood component-wise.

Transitions *form a ternary relation over $S \times \Sigma \times S$ and come in two types.*

Swaps: $(R, \theta, \hat{R}) \xrightarrow{s_u(\lambda)} (\lambda R, \theta, \hat{R}')$ *with $\psi(R) = \psi(\lambda R)$.*
Fractions θ of ownership are invariant under swaps (as new LP tokens are distributed proportionally). New net wealths of u and $v \neq u$ are respectively given by:

$$\begin{array}{rcl} \hat{R}'_u & = & \hat{R}_u + (1-\theta_u)(\mathbf{1}-\lambda)R \\ \hat{R}'_v & = & \hat{R}_v - \theta_v(\mathbf{1}-\lambda)R \end{array}$$

Transfers: $(R, \theta, \hat{R}) \xrightarrow{t_u(\mu)} (\mu R, \theta', \hat{R}')$
We set $\nu := \psi(\mu R)/\psi(R)$.
New ownership fractions and net wealths of u and $v \neq u$ are respectively given by:

$$\left\{\begin{array}{rcl} \theta'_u & = & \theta_u + (1-\theta_u)(1-\nu^{-1}) \\ \hat{R}'_u & = & \hat{R}_u + (1-\theta_u)(\mathbf{1}-\mu\nu^{-1})R \end{array}\right. \qquad \left\{\begin{array}{rcl} \theta'_v & = & \theta_v\nu^{-1} \\ \hat{R}'_v & = & \hat{R}_v - \theta_v(\mathbf{1}-\mu\nu^{-1})R \end{array}\right.$$

With this long definition in place, we can make a number of remarks.

As said, factors λ, μ in the (Swap) and (Transfer) transitions above are in general vectors, not scalars.

In (Swap) transitions, tokens received and paid out by the price machine can be written explicitly in multiplicative form:

$$(R' - R)_+ = (\lambda - \mathbf{1})_+ R$$
$$(R - R')_+ = (\mathbf{1} - \lambda)_+ R$$

The pre-factor $(1-\theta_u)$ occurring in u's post-swap net wealth $\hat{R}'_u$ expresses the fact that u is partly self-trading if s/he also holds LP tokens. This pre-factor can therefore be interpreted as a "wash trading" factor. In particular, if $\theta_u = 1$ (and therefore $\theta_v = 0$ for $v \neq u$) net wealths are invariant under swap; which is to be expected as, in this case, user u is entirely trading with self.

The total amount of tokens is conserved under any transition: $\mathbf{1}^T\hat{R} = \mathbf{1}^T\hat{R}'$.

Transitions are subject to a budget constraint, $\hat{R}'_u \succeq 0$ for u a user; ie user u holds non-negative amounts of tokens post-transition. For a Transfer the constraint can be written $\hat{R}'_u - \theta'_u R' \succeq 0$.

Finally, notice that $\hat{R} = \hat{R}'$ under perfectly balanced transfers (because in this case $\mu = \nu\mathbf{1}$). (This makes it convenient to work with net wealths and multiplicative transitions rather than local reserves and additive transitions.)

2.5 Single user price machines

In the following, we only need to consider the case of a single user u (the attacker) interacting with the price machine. Caveat: this does not mean that other users are not present as LPs, just that they do not interact with the machine while u is. In particular, u is allowed to have varying amounts of fraction of ownership of the pools. (A similar two-user version of a price machine is a convenient model to study the so-called MEV attacks on a price machine.)

▶ **Definition 2.** *In the single user case, the data presented in the preceding definition simplifies as follows.*

Single user states reduce to the simpler form $(R, \theta_u, \hat{R}_u) \in (\mathbb{R}^*_+)^n \times \mathbb{R}_+ \times \mathbb{R}^n_+$.

Labelled transitions simplify to:

Swaps:

$$(R, \theta_u, \hat{R}_u) \quad \xrightarrow{s_u(\lambda)} \quad (\lambda R, \theta_u, \hat{R}_u + (1-\theta_u)(\mathbf{1}-\lambda)R)$$

with $\psi(\lambda R) = \psi(R)$

Transfers:

$$(R, \theta_u, \hat{R}_u) \quad \xrightarrow{t_u(\mu)} \quad (\mu R, \theta_u + (1-\theta_u)(1-\nu^{-1}), \hat{R}_u + (1-\theta_u)(\mathbf{1}-\nu^{-1}\mu)R)$$

with $\nu := \psi(\mu R)/\psi(R)$

Let us verify that the transitions above are correct.

For a Swap $(R, \theta, \hat{R}_u) \xrightarrow{s_u(\lambda)} (\lambda R, \theta, \hat{R}'_u)$, $\theta = \theta_u$ is unchanged.

Local reserves become:

$$R'^{loc}_u := \hat{R}_u - \theta_u R - (\lambda - \mathbf{1})R$$

hence

$$\begin{aligned} \hat{R}'_u &:= R'^{loc}_u + \theta_u \lambda R \\ &= \hat{R}_u + (1-\theta_u)(\mathbf{1}-\lambda)R \end{aligned}$$

as in the expression given above.

For a Transfer $(R, \theta, \hat{R}_u) \xrightarrow{t_u(\mu)} (\mu R, \theta', \hat{R}'_u)$, the total amount of LP tokens post transition is $\psi(\mu R)$. The difference in LP tokens, $\psi(\mu R) - \psi(R)$ (negative for a withdraw), is given to u. Therefore the amount of LP tokens held by u after the transition is $\theta'_u\psi(\mu R) = \theta_u\psi(R) + \psi(\mu R) - \psi(R)$. Hence:

$$\theta'_u = \theta_u + (1-\theta_u)(1-\nu^{-1})$$

The amount given by u is $(\mu - \mathbf{1})R$, so $R'^{loc}_u = R^{loc}_u - (\mu - \mathbf{1})R$, and:

$$\begin{array}{rcl}\hat{R}'_u & = & R'^{loc}_u + \theta'_u \mu R \\ & = & R^{loc}_u - (\mu - \mathbf{1})R + (\theta_u + (1-\theta_u)(1-\nu^{-1}))\mu R \\ & = & \hat{R}_u - \theta_u R - (\mu - \mathbf{1})R + (\theta_u + (1-\theta_u)(1-\nu^{-1}))\mu R \\ & = & \hat{R}_u + (1-\theta_u)R - \mu R + (\theta_u + (1-\theta_u)(1-\nu^{-1}))\mu R\end{array}$$

and the expression given above for $\hat{R}'_u$ follows.

In the special case of a perfectly balanced transfer, $\nu^{-1}\mu = \mathbf{1}$ and $\hat{R}'_u = \hat{R}_u$, ie the net wealth of u is unchanged.

3 Consistency

Intuitively, a price machine is consistent if no attacker can extract price-independent profit, regardless of the machine's initial state. With the vocabulary developed in the preceding section, we can now formulate this precisely.

Given a price machine, a sequence of transitions caused by the same user u is called a *trace*. We say that two traces are *equivalent* if they have the same initial and final state. Clearly, equivalence is compatible with composition.

Equivalent traces may induce different budget constraints. (See an example below.)

Concretely, in a blockchain with a sequential execution model, the attacker can drive the price machine to execute any trace of his choice into one single transaction (that is to say atomically).

▶ **Definition 3** (Consistency). *A price machine is said to be consistent if, for any trace σ caused by u:*

$$(R, \theta_u, \hat{R}_u) \xrightarrow{\sigma} (R', \theta'_u, \hat{R}'_u)$$

$\hat{R}_u \preceq \hat{R}'_u$ *implies* $\hat{R}_u = \hat{R}'_u$.

As we will see below, it is easy to show that a trace which includes only Swap transitions cannot be a counter-example to consistency [2]. (Essentially because along such a sequence the invariant $\psi(R)$ cannot decrease.) The idea of the consistency proof is to reduce a trace into an equivalent one, where Swaps are all executed before any Transfer transition happens, and Transfers are perfectly balanced. Perfectly balanced transfers do not alter the user net wealth (as noticed a few lines above). So the conclusion would follow.

▶ **Proposition 4.** *Given a price machine with 1-homogenous trade function, an arbitrary transfer is equivalent to a swap (without fees) followed by a balanced transfer.*

Proof. We use reduced states and transitions (Def. 2). Consider an arbitrary transfer $t_u(\mu)$. Define $\nu := \psi(\mu R)/\psi(R)$.

We have the following trace:

$$\begin{array}{lll} R, \theta_u, \hat{R}_u & \xrightarrow{s_u(\nu^{-1}\mu)} & \nu^{-1}\mu R, \theta_u, \hat{R}_u + (1-\theta_u)(\mathbf{1} - \nu^{-1}\mu)R \\ & \xrightarrow{t_u(\nu\mathbf{1})} & \mu R, \theta_u + (1-\theta_u)(1-\nu_*^{-1}), \\ & & \hat{R}_u + (1-\theta_u)(\mathbf{1} - \nu^{-1}\mu)R + (1-\theta_u)(\mathbf{1} - \nu_*^{-1}\nu\mathbf{1})R \end{array}$$

with $\nu_* = \psi(\mu R)/\psi(\nu^{-1}\mu R) = \nu$, because ψ is 1-homogenous, and therefore, the above trace is equivalent to a direct Transfer $t_u(\mu)$, as can be read directly from Def. (2). We also have to check that the first transition is correct:

$$\psi(\nu^{-1}\mu R) \quad = \quad \psi(\psi(R)/\psi(\mu R)\,\mu R) \quad = \quad \psi(R)$$

again by 1-homogeneity of ψ. ◀

▶ **Proposition 5.** *Given a price machine with 1-homogenous trade function, swaps and balanced transfers due to the same user commute.*

Proof. Given a reduced state $(R, \theta_u, \hat{R}_u)$, we have

$$(R, \theta_u, \hat{R}_u) \xrightarrow{t_u(\mu)s_u(\lambda)} (\lambda\mu R, \theta_u + (1-\theta_u)(1-\nu^{-1}), \hat{R}_u + (1-\theta_u)(\mathbf{1} - \lambda\mu\nu^{-1})R),$$
$$(R, \theta_u, \hat{R}_u) \xrightarrow{s_u(\lambda)t_u(\mu)} (\lambda\mu R, \theta_u + (1-\theta_u)(1-\nu_*^{-1}), \hat{R}_u + (1-\theta_u)(\mathbf{1} - \lambda\mu\nu_*^{-1})R),$$

where $\nu = \psi(\mu R)/\psi(R) = \psi(\mu\lambda R)/\psi(\lambda R) = \nu_*$, with the middle equality because ψ is 1-homogenous. ◀

The "commutation" above has to be understood up to budget constraints. Indeed, if we consider the constant product ψ (see §2.3), with initial state $(R, \theta_u, \hat{R}_u)$, where $R = (60, 60)$, $\theta_u = 0.5$, $\hat{R}_u = (80, 80)$, and transitions $t_u(\mu)$, $s_u(\lambda)$, are defined by $\mu = (0.5, 0.5)$ (balanced), and $\lambda = (0.5, 2)$. The budget constraint prevents the expected equivalence between $t_u(\mu)s_u(\lambda)$ and $s_u(\lambda)t_u(\mu)$.

We can now wrap up the proof.

▶ **Theorem 6.** *A price machine is consistent if its trading function is* 1*-homogenous.*

Proof. Given a state $(R, \theta_u, \hat{R}_u)$ and an attack trace σ. Since the feasibility of swaps and transfers is preserved under a positive translation on $\hat{R}_u$, we may suppose that $\hat{R}_u$ is large enough (the attacker has deep pockets).

By Prop. 4 (decomposition of non-balanced transfers), σ is equivalent to a trace σ_1 where every transfer is balanced.

By Prop. 5 (postponement of transfers), σ_1 is in turn equivalent to a trace σ_2 of the form $s_u(\lambda_1) \cdots s_u(\lambda_n) t_u(\mu_1) \cdots t_u(\mu_m)$, where each $t_u(\mu_i)$ is balanced.

It is easy to see that sequences of single-user balanced transfers can be aggregated, meaning $t_u(\mu_1) \cdots t_u(\mu_m)$ is equivalent to a one step transfer $t_u(\mu)$ with $\mu = \mu_1 \cdots \mu_m$. Likewise, sequences of single-user swaps $s_u(\lambda_1) \cdots s_u(\lambda_n)$ are equivalent to a one step swap $s_u(\lambda)$ with $\lambda = \lambda_1 \cdots \lambda_n$.

By combining both remarks we can obtain a trace σ_3 equivalent to the original σ and of the simple form $s_u(\lambda)t_u(\mu)$.

Let now $\hat{R}'_u$ be the net wealth of user u at the end of σ_3 (equivalently at the end of σ). Since $t_u(\mu)$ is perfectly balanced, $\hat{R}'_u$ is also the net wealth of u after the combined swap $s_u(\lambda)$. Hence $\hat{R}'_u = \hat{R}_u + (1-\theta_u)(\mathbf{1} - \lambda)R$.

Now suppose that $\hat{R}_u \preceq \hat{R}'_u$, it must be that $\lambda \preceq \mathbf{1}$. Since $\psi(R) = \psi(\lambda R)$ (no fees) and ψ is (strictly) increasing, it must be in fact that $\lambda = \mathbf{1}$. Hence $\hat{R}_u = \hat{R}'_u$. ◀

We do not really need to suppose that swaps have zero fees. The proof above can handle the case where $s_u(\lambda_1) \cdots s_u(\lambda_n) t_u(\mu)$ consists of swaps with and without fees.

4 Applications

It remains to show that our approach applies to some interesting AMMs.

Recall that the epigraph and the hypograph of a function $f : X \to \overline{\mathbb{R}}$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$, are defined as:

$$\text{epi}(f) := \{(x, r) \in X \times \mathbb{R} : r \geq f(x)\},$$
$$\text{hyp}(f) := \{(x, r) \in X \times \mathbb{R} : r \leq f(x)\}.$$

To do this we define a specific class of candidate trading functions.

▶ **Definition 7.** *Let a triple* (f, g, F) *be given with:*

- $f,\ g : (\mathbb{R}_+^*)^n \to \overline{\mathbb{R}}_+$
- $f \leq g$
- $F : \operatorname{epi} f \cap \operatorname{hyp} g \to \mathbb{R}$

Suppose that for all $R \in (\mathbb{R}_+^*)^n$*, there is a unique* D *such that* $F(R, D) = 0$ *and* $f(R) \leq D \leq g(R)$*. The candidate trading function associated to* (f, g, F) *is then defined as* $\psi(R) := D$*.*

Intuitively, $f(R)$, $g(R)$ give bounds on the amount of LP tokens $\psi(R)$ available for a given level of reserves R.

Now we ask for sufficient conditions for such a function to be increasing, i.e. to be an actual trading function.

▶ **Theorem 8.** *Let* ψ *be as in Def. 7. For* ψ *to be increasing, it is sufficient that the following holds:*

1. $\forall R \in \operatorname{dom}\psi$, $F(R, \cdot)$ *is of class* $\mathcal{C}^1$ *on* $[f(R), g(R)]$, *with* $F'(R, \cdot) < 0$
2. $\forall D \in \operatorname{im}\psi$, $F(\cdot, D)$ *is strictly increasing*
3. f, g *are continuous, and,* $f < \psi < g$ *almost everywhere*

Proof. From condition 1, we know that ψ is of class $\mathcal{C}^1$, by the inverse function theorem. Since we also have condition 3, it suffices to prove that ψ is locally strictly increasing, for every R such that $f(R) < \psi(R) < g(R)$.

Now, pick $R \in \operatorname{dom}\psi$ such that $f(R) < \psi(R) < g(R)$. By condition 3 and the continuity of ψ, there is an open neighbourhood N_R of R such that for almost every $R' \in N_R$, both $f(R') < \psi(R') < g(R')$ and $f(R) < \psi(R') < g(R)$ hold.

Suppose, without loss of generality, that $R \prec R'$. By definition, we have $F(R, \psi(R)) = F(R', \psi(R')) = 0$. By condition 2, we have $F(R, \psi(R')) < F(R', \psi(R'))$, so that $F(R, \psi(R)) > F(R, \psi(R'))$. By condition 1, we have $\psi(R) < \psi(R')$. Done! ◀

▶ **Theorem 9.** *Let* ψ *be a trading function defined as in Def. 7 via a triple* F, f, g*. Suppose* F *is positively homogenous, and* f, g *are 1-homogenous. Then* ψ *is positively homogenous of degree 1.*

Proof. Suppose that F is positively homogenous of degree k. Pick $R \in \operatorname{dom}\psi$, and $s > 0$. We have $F(sR, s\psi(R)) = s^k F(R, \psi(R)) = 0$, therefore $F(sR, s\psi(R)) = 0$. On the other hand, by definition $\psi(R) \geq f(R)$, hence $s\psi(R) \geq sf(R) = f(sR)$. Similarly $\psi(R) \leq g(R)$, hence $s\psi(R) \leq sg(R) = g(sR)$ Therefore by unicity of the solution to $F(sR, \_) = 0$ in the interval $[f(sR), g(sR)]$, it must be that $s\psi(R) = \psi(sR)$. ◀

4.1 The Balancer family

Balancer v1 is a multi-token AMM that generalizes [7] Uniswap v2's [1]. It is one of the formulas supported by Balancer v2. The trading function for Balancer with n tokens, denoted by ψ_B, is defined as in the general framework, with $f := 0$, $g := +\infty$, and

$$F(R, D) := \prod_{i=1}^{n} R_i^{w_i} - D,$$

where the weights in $w \in (\mathbb{R}_+^*)^n$ satisfy $\mathbf{1}^T w = 1$.

The trading function for Uniswap v2 (seen above) can be seen as a special case of that of Balancer, with $n = 2$ and $w_1 = w_2 = 0.5$.

▶ **Proposition 10.** ψ_B *is increasing and positively homogenous of degree* 1.

Proof. Clear. We even have $\psi_B(R) := \prod_{i=1}^{n} R_i^{w_i}$. ◀

▶ **Corollary 11.** *The Balancer family is consistent.*

We can remark that the on-chain implementation of a non-balanced deposit $R \rightarrow^t R'$ actually only returns $\tau := \min_i(R' - R)$ LP tokens [9], so that the balanced deposit $R \rightarrow \tau R$ is already more advantageous. A *reasonable* user always makes balanced transfers.

4.2 Application: the Curve family

Curve v1 is a multi-token AMM that offers arguably fairer trades than Uniswap v2 [5]. It is also one of the formulas supported by Balancer v2. Curve v2 is based on the same idea as Curve v1, but with some notable changes [6]. In these AMMs, transfers do not need to be balanced.

The trading function(s) for Curve with n tokens, denoted by ψ_C, is defined as in the general framework:

- $f(R) := n(\prod_{i=1}^{n} R_i)^{\frac{1}{n}} \leq \mathbf{1}^T R =: g(R)$
- with

$$F(R,D) \quad := \quad D^n \left[K(R,D)\left(g(R)D^{-1} - 1\right) + n^{-n}\left(\left(f(R)D^{-1}\right)^n - 1\right)\right]$$

where $K(R,D)$ is defined as either of:

$$\begin{aligned} K(R,D) \quad &:= \quad A(f(R)D^{-1})^n && \textit{(Curve v1)} \\ &:= \quad A(f(R)D^{-1})^n \frac{\gamma^2}{(\gamma+1-(f(R)D^{-1})^n)^2} && \textit{(Curve v2)} \end{aligned}$$

and $A \geq 0$ is called the *amplification coefficient*, and $\gamma > 0$ is a small constant.

▶ **Proposition 12.** *F is positively homogenous of degree n.*

Proof. Clear, since f, g are 1-homogenous, and K is 0-homogenous. ◀

▶ **Proposition 13.** *Condition 1 is verified.*

Proof. It's not hard to check that $\forall D \in \pi_{n+1} \circ \operatorname{dom} F$, for Curve v1,

$$F'(R,\cdot)(D) \quad = \quad -D^{n-1}n^{-n+1} - D^{-2}Af(R)^n g(R) \quad < \quad 0$$

and, for Curve v2, by denoting $T(R,D) := \gamma + 1 - (f(R)D^{-1})^n > 0$,

$$\begin{aligned} F'(R,\cdot)(D) \quad &= \quad -D^{n-1}n^{-n+1} - D^{-2}(f(R))^n g(R)A\gamma^2 T(R,D)^{-2} \\ &\quad\;\; -2D^{-n-2}(f(R))^{2n}(g(R)-D)An\gamma^2 T(R,D)^{-3} \\ &< \quad -D^{n-1}n^{-n+1} \qquad\qquad\qquad\qquad\qquad < \quad 0, \end{aligned}$$

so that condition 1 is verified. ◀

▶ **Proposition 14.** *ψ_C is well-defined. In addition, condition 3 is verified.*

Proof. We have $\forall R \in (\mathbb{R}_+^*)^n$, $F(R, g(R)) \leq 0 \leq F(R, f(R))$, so that ψ_C is well-defined, by the arithmetic-geometric mean inequality. Since the equalities hold if and only if $R_1^{-1}R = \mathbf{1}$, condition 3 is verified. ◀

It is worth noting that for Curve v1, F can be analytically continued to $(\mathbb{R}_+^*)^n \times \mathbb{R}_+^*$, where $\forall R \in (\mathbb{R}_+^*)^n$, $F(R, 0_+) > 0$, and $F'(R,\cdot)$ has at most one stationary point, so that $\psi_C(R)$ is in fact the unique solution of $F(R,\cdot) = 0$ on $\mathbb{R}_+^*$.

▶ **Proposition 15.** *Condition 2 is verified.*

Proof. Clearly, it suffices to prove that $\forall D \in \operatorname{im}\psi$, $K(\cdot, D)$ is strictly increasing. This is clear for Curve v1, but also for Curve v2, by the fact that $(fD^{-1})^n \leq 1 \leq \gamma + 1$, and that $(fD^{-1})^n$ is strictly increasing on $\operatorname{dom} K(\cdot, D)$. ◀

▶ **Corollary 16.** *ψ_C is strictly increasing and homogenous of degree 1.*

▶ **Corollary 17.** *The Curve family is consistent.*

5 Conclusion

We have proved the consistency of a specific family of (zero-fees) AMMs which are based on increasing and 1-homogenous invariants (also called trading functions). This is a new result. Note again that consistency says nothing of an AMM's suitability or efficiency as a market mechanism, just that it is not outright flawed.

Some generalisations can be expected. For example, while it is fairly intuitive that the consistency of AMMs with fees follows from that without, it remains to be proven rigorously.

One way to use our result is in the course of designing new AMMs. For instance, it follows directly from our result that the trading function $\psi(X, Y) = (X^3Y + XY^3)^{1/4}$ which has been proposed recently generates a consistent AMM. In the same vein, one could generalise the Curve approach of mixing two existing price machines (in the specific Curve construction one mixes the linear $X + Y$ invariant and the product XY one) to obtain a general consistency-preserving mixing combinator on the space of price machines.

However, note that our method only offers a sufficient condition. A typical example that does not fall under our result is the non-homogenous trading function $\psi(X, Y) = X + Y + XY$. It can be shown independently that this particular choice is indeed inconsistent in the sense of Def. 3 and can be exploited, but the results obtained in this paper do not give us a specific method to look for such an attack.

The reader might wonder if our approach applies also to the Uniswap v3 protocol. Uniswap v3 is not exactly an AMM but rather a (n efficient) aggregator of AMMs each based on a concentrated version of the original (Uniswap v2) product invariant. In general, in the case of protocols aggregating AMMs, the consistency question boils down to the consistency of the individual AMMs being aggregated. Now in Uniswap v3 every LP-position has a single liquidity provider, hence transfers are trivial ($\theta_u = 1$ at all times in the language of Section 2) and consistency follows from the monotony of the (concentrated) product invariant.

References

1 Hayden Adams, Noah Zinsmeister, and Dan Robinson. *Uniswap v2 Core*, 2020.

2 Guillermo Angeris and Tarun Chitra. Improved price oracles. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. ACM, 2020. `doi:10.1145/3419614.3423251`.

3 CoinMarketCap. Top cryptocurrency decentralized exchanges ranked, 2022. URL: `https://coinmarketcap.com/rankings/exchanges/dex/`.

4 CryptoSec. Comprehensive list of defi hacks and exploits, 2022. URL: `https://cryptosec.info/defi-hacks/`.

5 Michael Egorov. Stableswap-efficient mechanism for stablecoin liquidity. *Retrieved Feb*, 24:2021, 2019.

6 Michael Egorov. Automatic market-making with dynamic peg. *Retrieved Dec 2021*, June 2021. URL: `https://curve.fi/files/crypto-pools-paper.pdf`.

7 Fernando Martinelli and Nikolai Mushegian. *Balancer: A non-custodial portfolio manager, liquidity provider, and price sensor*, 2019.

8 Nick Szabo. Formalizing and securing relationships on public networks, September 1997. URL: `https://firstmonday.org/ojs/index.php/fm/article/view/548`.

9 Uniswap. Core smart contracts of Uniswap v2, 2020. URL: `https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol`.

10 Chiqing Zhang. Test cases implementing the example, 2022. retrieved Sep 2022. URL: `https://github.com/zhangchiqing/v2-periphery/blob/master/test/WalkThrough.spec.ts`.

Interest Rate Rules in Decentralized Finance: Evidence from Compound

Amit Chaudhary
Warwick Business School, University of Warwick, Coventry, UK

Roman Kozhan
Warwick Business School, University of Warwick, Coventry, UK

Ganesh Viswanath-Natraj
Warwick Business School, University of Warwick, Coventry, UK

Abstract

We study the fundamentals of interest rate rules on the decentralized finance protocol Compound. Interest rates are set by the governance of the protocol, and are based on the utilization of an asset: which is the ratio of a cryptocurrency that is borrowed to its total supply in the protocol. We discuss factors that determine the slope parameters of interest rate rules. Slope parameters are typically higher for more volatile cryptocurrencies. We argue liquidation risk can explain the cross-sectional variation in interest rate rules. We also draw parallels between these rules to the demand for loanable funds in traditional money markets.

2012 ACM Subject Classification Applied computing

Keywords and phrases Cryptocurrency, decentralized finance lending protocols, monetary policy, stablecoins, governance token

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.5

1 Introduction and Motivation

DeFi is a blockchain based form of finance that runs on smart contracts and does not need a centralized financial intermediary, such as a market maker or a bank. Lending platforms running on the Ethereum blockchain, such as Compound, set interest rates and allocate funds automatically through algorithms. They allow users to deposit multiple collateral types, for example ETH and Wrapped Bitcoin (WBTC), and borrow multiple currencies, such as stablecoins like DAI and USDC. Benefits include the instantaneous settlement of contracts, and minimizing counterparty risk for lenders, so reducing the likelihood that any party involved will default. This is done through a system of smart contracts where borrowers are required to post sufficient collateral.

A unique feature of decentralized lending protocols is that interest rates are determined algorithmically by the utilization in the market, which is calculated as the ratio of total borrowing to total supply. This is in contrast to traditional money markets where the interest rate is typically set exogenously by monetary policy. In this paper we will discuss interest rate rules on the Compound protocol. We will show that parameters governing the rule, such as the sensitvity of interest rates to utilization, are typically higher for more volatile cryptocurrencies. We also draw parallels between these interest rate models and models of demand for loanable funds in traditional money markets.

Empirical work on lending protocols has focused on understanding market efficiency, such as uncovered interest rate parity, the behavior of liquidations during risk-off events, the dynamics of the COMP governance token, and theoretical work on the stability of interest rate rules [4, 8, 11, 3, 6, 1, 9, 14, 7, 2, 13, 10]. Within this literature, our paper shows that interest rate rules differ between risky cryptocurrencies and stablecoins.

In section 2 we outline interest rate rules on Compound. Section 3 discusses sources of risk that explain the interest rate rule parameters, and compares the algorithmic setting of interest rates to traditional money markets. Section 4 concludes.

2 Interest rate rules

Collateralized lending markets like Compound allow users to borrow and lend in multiple currencies by tapping into liquidity pools of multiple assets. Users supply a collateral asset, and can borrow a fraction as tokens in another asset that is based on the collateral factor of a given asset. Each market has separate interest rate curves on borrowing and lending that is based on the relative utilization (ratio of borrowing to lending) of that asset. The supply and borrow interest rates are compounded every block. [1]

One key feature of governance is to vote on interest rate rules.[2] Parameters like the base-rate and slope of the interest rate model are chosen by voters as part of the governance protocol. The interest rate model for borrowing rates is given by the piece-wise equation (1).[3] a_0 is the base rate, and is the rate corresponding to zero utilization. The slope parameter $b_0 > 0$ measures the sensitivity of interest rates to utilization. The utilization rate u is used as an input parameter to a formula that determines the interest rates. Interest rates are determined by the utilization percentage in the market. Utilization is calculated as the ratio of total borrowings to total supply of the asset. All else equal, a positive slope parameter implies higher utilization leads to higher interest rates. An additional feature of the interest rate model is the kink, in which the slope parameter changes for utilization above a threshold rate $\bar{u}$, typically 80 per cent. The kink makes interest rates more sensitive to a higher utilization rate, $b_1 > b_0$.

$$i_L = \begin{cases} a_0 + b_0 u, u \leq \bar{u} \\ a_0 + b_0 \bar{u} + b_1(u - \bar{u}), u > \bar{u} \end{cases} \tag{1}$$

Deposit rates i_D is a function of utilization and borrowing rates. θ captures the fraction of interest income that is in a reserve buffer managed by the interest rate protocol.

$$i_D = u i_L (1 - \theta) \tag{2}$$

In Panel A of Figure 1, we plot the utilization percentage of the 6 principal cryptocurrencies offered in the Compound protocol. The sample period is from August 5th 2020 to October 22nd, 2022. Utilization rates of stablecoins (USDC, Tether and DAI) are systematically higher than risky cryptocurrencies (ETH, ZRX and WBTC). In Panel B, we plot the interest rate model for currencies on the Compound platform. Interest rate rules for more volatile cryptocurrencies typically have a higher slope parameter. Figure 2 estimates the slope parameters for each currency. Stablecoins have an average slope parameter of approximately 0.07 to 0.10. In contrast, risky cryptocurrencies like WBTC and ZRX have an average slope

[1] Blocks are measured approximately 15 seconds on Ethereum therefore producing approximately continuous compounding.

[2] Governance token COMP used to vote on interest rate rule parameters. To create a proposal a user requires at least 100,000 COMP tokens. A user with 100 COMP can initiate a proposal but require community to support through delegating tokens. All proposals are first discussed publicly in an official governance forum, are written in smart contracts.

[3] For a discussion of alternative interest rate models on other protocols like Aave, we refer readers to [4]. While the functional forms may vary, other protocols typically posit interest rates as a positive function of utilization.

parameter of approximately 0.3. Therefore a 1 per cent increase in utilization raises interest rates of stablecoins by 7 to 10 basis points, and risky cryptocurrencies by 30 basis points, all else equal. We now turn to factors that determine the cross-sectional variation in interest rate rule parameters.

3 Discussion

Interest rate rules for more risky assets typically have higher slope parameters. Parameters like the slope of the interest rate model are chosen by voters as part of the governance protocol. While higher utilization rates will increase net interest income to the protocol, it will also increase systemic risks in the protocol, such as liquidations, which are when individuals become over-leveraged and borrow too much relative to the collateral they post on the protocol. Liquidation risk has systemic effects in the pricing of collateral assets [6, 3, 13]. If liquidations trigger fire sales of collateral, and the protocol does not have sufficient net interest income, the protocol will mint governance tokens (COMP) to repay the debt. This will result in a devaluation of the governance token and incur losses for users of the protocol.[4]

Second, the interest rate schedule in equation (1) shows that slope parameters are higher after a threshold rate of utilization. This feature has parallels to models for excess reserves in traditional money markets. In [12], the authors find that in money markets the interest rate schedule becomes steeper when excess reserves are smaller, and model excess reserve balances with a logistic function. When utilization is high, excess reserves of the protocol, which we define as the difference between the collateral supplied and the amount that is borrowed, diminishes. Therefore, the non-linear kink in the Compound interest rate model helps preserve excess reserves in the protocol.

4 Conclusion

DeFi lending platforms like Compound, running on Ethereum, allow users to deposit various types of collateral and borrow multiple currencies. Smart contracts help minimize counterparty risks, and interest rates are determined algorithmically by market utilization. The parameters of the interest rate rule, such as the slope parameter, are typically higher for more volatile cryptocurrencies. Differences in interest rate rules are due to the role of liquidation risk. The interest rate model parallels traditional money markets' demand for loanable funds.

References

1 Carlos Castro-Iragorri, Julian Ramirez, and Sebastian Velez. Financial intermediation and risk in decentralized lending protocols. *Available at SSRN 3893278*, 2021.

2 Amit Chaudhary and Daniele Pinna. Market risk assessment: A multi-asset, agent-based approach applied to the 0vix lending protocol. *arXiv preprint*, 2022. `arXiv:2211.08870`.

3 Jonathan Chiu, Emre Ozdenoren, Kathy Yuan, and Shengxing Zhang. The fragility of defi lending, 2022.

4 Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020.

[4] Examples of liquidations leading to a devaluation of the governance token occurred for MakerDAO's DAI protocol. During the *Black Thursday Crypto crash* on March 12th 2020, MKR governance tokens were minted to pay off the DAI debt triggered by liquidations. For more details we refer readers to [5].

5 Roman Kozhan and Ganesh Viswanath-Natraj. Fundamentals of the MakerDAO governance token. In *3rd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/OASIcs.Tokenomics.2021.11`.

6 Alfred Lehar and Christine A Parlour. Systemic fragility in decentralized markets, 2022.

7 Peter Mueller. Defi leveraged trading: Inequitable costs of decentralization. *Available at SSRN 4241356*, 2022.

8 Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: DeFi on a Knife-edge. *arXiv preprint*, 2020. `arXiv:2009.13235`.

9 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021.

10 Thomas J Rivera, Fahad Saleh, and Quentin Vandeweyer. Equilibrium in a DeFi lending market. *Available at SSRN 4389890*, 2023.

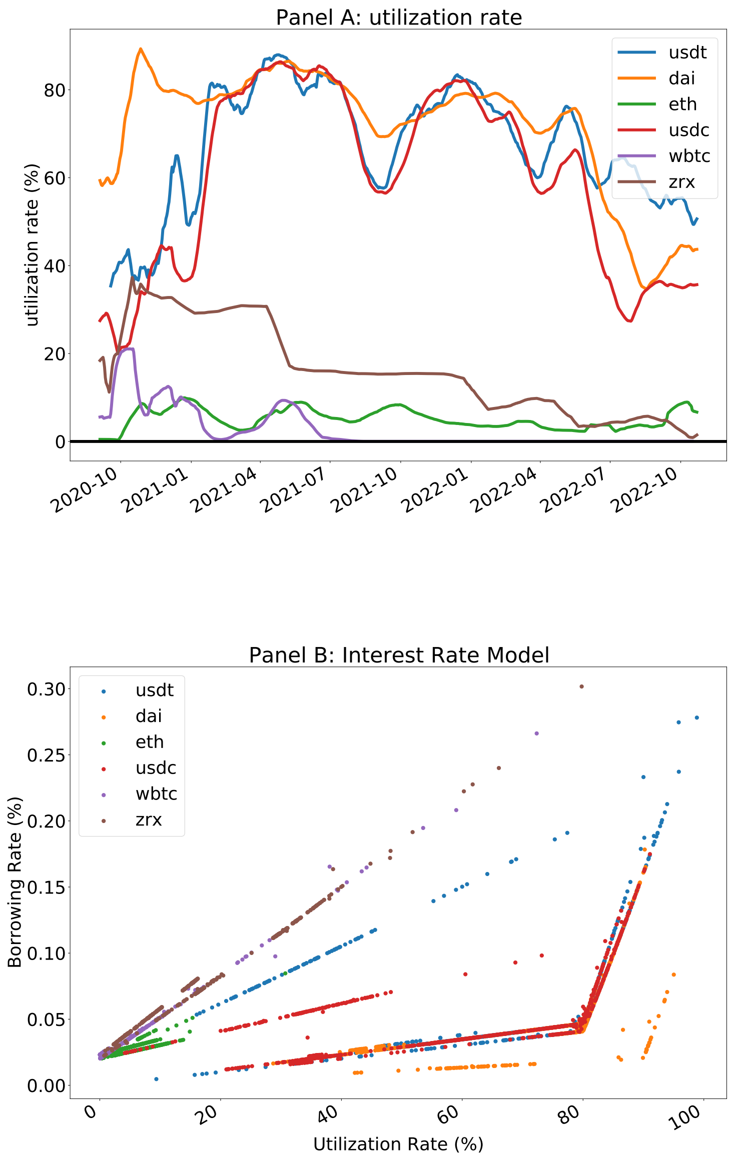
11 Kanis Saengchote. Decentralized lending and its users: Insights from compound. *Available at SSRN 3925344*, 2021.

12 Mr Romain M Veyrune, Guido Della Valle, and Shaoyu Guo. *Relationship Between Short-Term Interest Rates and Excess Reserves: A Logistic Approach.* International Monetary Fund, 2018.

13 Jakub Warmuz, Amit Chaudhary, and Daniele Pinna. Toxic liquidation spirals: Evidence from the bad debt incurred by AAVE. *arXiv preprint*, 2022. `arXiv:2212.07306`.

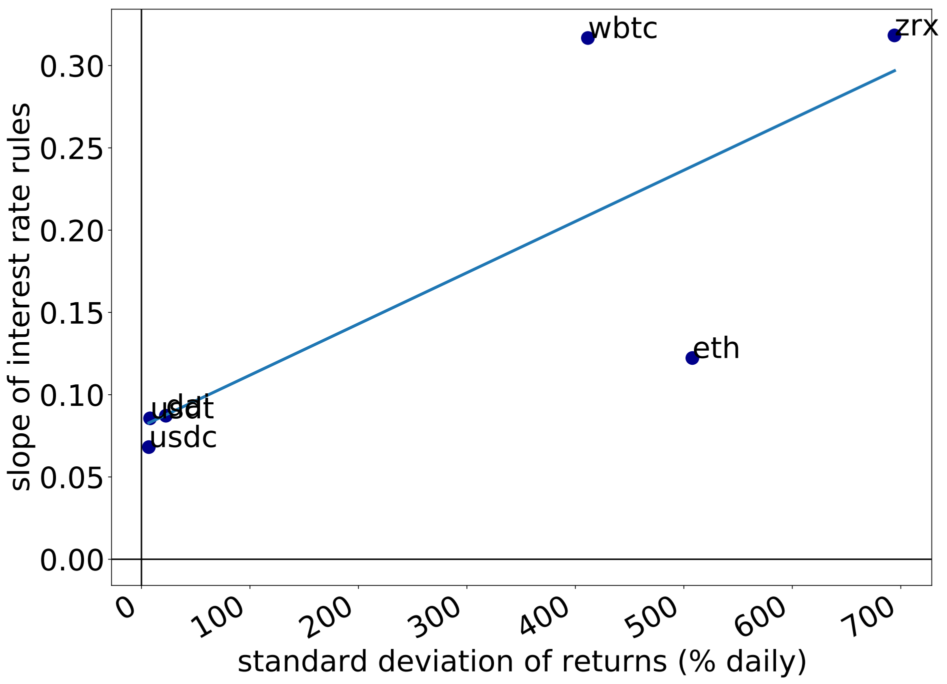
14 Jiahua Xu and Nikhil Vadgama. From banks to defi: the evolution of the lending market. In *Enabling the Internet of Value*, pages 53–66. Springer, 2022.

A Appendix



Note: Figure top panel presents utilization rates (in percentage points) on multiple assets, calculated as a historical rolling average over the last 30 days. Bottom panel plots interest rate models on multiple assets, in which borrowing rates are determined as a function of the utilization rate, Source: Compound API. Sample period is from August 5th, 2020 to October 22nd, 2022.

Figure 1 Utilization Rate and Interest Rate Rules.



Note: Figure plots slope of interest rate rules as a function of the standard deviation of daily returns. Source: Compound API and Cryptocompare. Sample period is from August 5th, 2020 to October 22nd, 2022.

Figure 2 Slope parameters of interest rate rules and cryptocurrency volatility.

Maximal Extractable Value (MEV) Protection on a DAG

Dahlia Malkhi
Chainlink Labs, UK

Pawel Szalachowski
Chainlink Labs, UK

Abstract

Many cryptocurrency platforms are vulnerable to Maximal Extractable Value (MEV) attacks [12], where a malicious consensus leader can inject transactions or change the order of user transactions to maximize its profit.

A promising line of research in MEV mitigation is to enhance the Byzantine fault tolerance (BFT) consensus core of blockchains by new functionalities, like hiding transaction contents, such that malicious parties cannot analyze and exploit them until they are ordered. An orthogonal line of research demonstrates excellent performance for BFT protocols designed around Directed Acyclic Graphs (DAG). They provide high throughput by keeping high network utilization, decoupling transactions' dissemination from their metadata ordering, and encoding consensus logic efficiently over a DAG representing a causal ordering of disseminated messages.

This paper explains how to combine these two advances. It introduces a DAG-based protocol called Fino, that integrates MEV-resistance features into DAG-based BFT without delaying the steady spreading of transactions by the DAG transport and with zero message overhead. The scheme operates without complex secret share verifiability or recoverability, and avoids costly threshold encryption.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases DAG, MEV, consensus, BFT

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.6

Acknowledgements We are grateful to Soumya Basu, Christian Cachin, Ari Juels, Mahimna Kelkar, Lefteris Kokoris-Kogias, Oded Naor, Mike Reiter for many comments that helped improve this writeup.

1 Introduction

1.1 MEV

Over the last few years, we have seen exploding interest in cryptocurrency platforms and applications built upon them, like decentralized finance protocols offering censorship-resistant and open access to financial instruments; or non-fungible tokens. Many of these systems are vulnerable to MEV attacks, where a malicious consensus leader can inject transactions or change the order of user transactions to maximize its profit. Thus it is not surprising that at the same time we have witnessed rising phenomena of MEV professionalization, where an entire ecosystem of MEV exploitation, comprising of MEV opportunity "searchers" and collaborating miners, has arisen.

Daian et al. [12] introduced a measure of the "profit that can be made through including, excluding, or re-ordering transactions within blocks". The original work called the measure miner extractable value, which was later extended by maximal extractable value (MEV) [33] and blockchain extractable value (BEV) [36], to include other forms of attacks, not necessarily performed by miners. At the time of this writing, an "MEV-explore" tool [1] estimates the amount of MEV extracted on Ethereum since the 1st of Jan 2020 to be close to $700M.

However, it is safe to assume that the total MEV extracted is much higher, since MEV-explore limits its estimates to only one blockchain, a few protocols, and a limited number of detectable MEV techniques. Although it is difficult to argue that all MEV is "bad" (e.g., market arbitrage can remove market inefficiencies), it usually introduces some negative externalities like: *network congestion*: especially on low-cost chains, MEV actors often try to increase their chances of exploiting an MEV opportunity by sending a lot of redundant transactions, spamming the underlying peer-to-peer network; *chain congestion*: many such transactions finally make it to the chain, making the chain more congested; *higher blockchain costs*: while competing for profitable MEV opportunities, MEV actors bid higher gas prices to prioritize their transactions, which results in overall higher blockchain costs for regular users; *consensus stability*: some on-chain transactions can create such a lucrative MEV opportunity that it may be tempting for miner(s) to create an alternative chain fork with such a transaction extracted by them, which introduces consensus instability risks.

1.2 Blind Order-Fairness on a DAG

A promising way of thwarting some forms of MEV on blockchains is to extend their BFT (Byzantine fault tolerance) consensus core by new properties like Blind Order-Fairness, where no consensus "validator" can learn the content of transactions until they are committed to a total ordering. This notion of *"commit-reveal"* is the key mechanism we focus on in this paper.

An orthogonal line of research concentrates on scaling the BFT consensus of blockchains via a *DAG communication substrate.* A DAG transport can spread yet-unconfirmed transactions in parallel, every message having utility and carrying transactions that eventually become committed to a total ordering. The DAG provides Reliability, Non-equivocation, and Causal-ordering delivery guarantees, enabling a simple and "zero-overhead" consensus ordering on top of it. That is, validators can interpret their DAG locally without exchanging more messages and determine a total ordering of accumulated transactions. Several recent DAG-based BFT systems, including Blockmania [13], Aleph [19], Narwhal [14], DAG-Rider [23], and Bullshark [20], demonstrate excellent performance.

This paper combines these two advances in a solution called Fino, that integrates smoothly two ingredients: A framework for commit-reveal consensus protocols, and a simple DAG-based BFT consensus that leverages the Reliability, Non-Equivocation, and Causality properties of the DAG transport to integrate the commit-reveal framework smoothly and efficiently into consensus.

A commit-reveal framework. The core of the commit-reveal framework of Fino is a standard k-of-n secret sharing approach consisting of two functionalities, Disperse() and Retrieve(). Users first send transactions to validators *encrypted*, so that the consensus protocol commits to an ordering of transactions *blindly.* Disperse() entrusts shares of the secret decryption key for each transaction to validators. After an ordering is committed, Retrieve() opens encrypted transactions by collecting shares from $F + 1$ validators.

Implementing Disperse()/Retrieve() efficiently enough to meet DAG-based consensus throughput is the core challenge Fino addresses. Two straw man approaches are (i) threshold cryptography, but it would incur an order of milliseconds per transaction, (ii) using polynomial secret-sharing and verifying shares during dispersal, also requiring costly cryptography as well as implementing a share-recovery protocol.

The third approach, and the one we recommend using, shares transaction encryption keys using Shamir's secret-sharing scheme [38], but without secret share verifiability or recoverability (slow and complex), and without threshold encryption (slow). Instead, to guarantee a unique, deterministic outcome of encrypted transactions, Fino borrows a key insight from DispersedLedger [40] called AVID-M, protecting against share manipulation attacks by verifying that the sharing was correct *post reconstruction*. If the dispersal was incorrect and fails post-reconstruction verification, the transaction is rejected.

Notably, this approach works in microseconds latency and is orders of magnitude faster than threshold encryption.

section 3 describes a fourth approach, a hybrid combining AVID-M with threshold cryptography. Hybrid tackles adversarial conditions where Disperse() partially fails, at the cost of a slower post-reconstruction verification.

We repeat that all four approaches above are compatible with the Fino framework, though the fastest and simplest one is AVID-M.

DAG-based integration. Fino integrates Disperse()/Retrieve() into a DAG-based consensus without delaying the steady spreading of transactions by the DAG transport and with zero message overhead. Validators interpret their local DAGs to arrive at commit decisions on blind ordering. After an ordering has committed for a batch of transactions, validators invoke Retrieve() and present decryption shares piggybacked on DAG broadcasts. Finally, validators interpret their local DAGs to arrive at unanimous, deterministic outcome of transaction opening.

During periods of synchrony, Fino has a happy path for transaction commit of two DAG latencies, plus a single DAG latency for determining an opening outcome.

2 Background and Preliminaries

2.1 System Model

The Byzantine fault tolerance (BFT) Consensus core of blockchains implements state machine replication (SMR), where non-faulty parties–also known as validators–agree on an order of transactions and execute them consistently. The goal of our work is to extend BFT consensus and build into it protection against MEV attacks which rely on transaction content analysis.

We assume that the system should be resilient to Byzantine faults, i.e., faulty validators can implement any adversarial behavior (like crashing, equivocating, or colluding). We assume a partially synchronous network [18], where asynchronous periods last up to unknown global stabilization time (GST), while in synchronous periods there is a known bound Δ on delays for message delivery. This model requires a BFT threshold $N \geq 3F + 1$, where N and F is the number of all and Byzantine faulty validators, respectively.

2.2 MEV and Blind Order-Fairness

In this paper, we focus on consensus-level MEV mitigation techniques. There are fundamentally two types of MEV-resistant Order-Fairness properties:

Blind Order-Fairness. A principal line of defense against MEV stems from committing to transaction ordering without seeing transaction contents. This notion of MEV resistance, referred to here as Blind Order-Fairness, is used in a recent SoK on Preventing Transaction Reordering by Heimbach and Wattenhofer [21], and is defined as: *"when it is not possible for any party to include or exclude transactions after seeing their contents. Further, it should not be possible for any party to insert their own transaction before any transaction whose contents it already been observed."*

Time-Based Order-Fairness. Another measure for MEV protection is brought by sending transactions to all BFT parties simultaneously and using the relative arrival order at a majority of the parties to determine the final ordering. In particular, this notion of order fairness ensures that *"if sufficiently many parties receive a transaction tx before another tx', then in the final commit order tx' is not sequenced before tx."*

This defines the transactions order and prevents adversaries that can analyze network traffic and transaction contents from reordering, censoring, and front-/back-running transactions received by Consensus parties. Moreover, Time-Based Order-Fairness protects against a potential collusion between users and BFT leaders/parties because parties explicitly input relative ordering into the protocol. Time-Based Order-Fairness is used in various flavors in several recent works, including Pompe [43], Aequitas [26], Themis [25], "Wendy Grows Up" [28], and "Quick Order Fairness" [11]. We briefly discuss some of those protocols in section 8.

Another notion of fairness found in the literature, that does not provide Order-Fairness, revolves around participation fairness:

Participation Fairness. A different notion of fairness aims to ensure censorship-resistance or stronger notions of participation equity. Participation Fairness guarantees that the committed sequence of transactions includes a certain portion of honest contribution (aka "Chain Quality"). Several BFT protocols address Participation Fairness, including Prime [2], Fairledger [29], HoneyBadger [31]. As mentioned in section 4, some DAG-based BFT protocols like Aleph [19], DAG-Rider [23], Tusk [14], and Bullshark [20] use a layered DAG paradigm. In this approach, Participation Fairness comes essentially for free because every DAG layer must include messages from 2F+1 participants. It is worth noting that Participation Fairness does not prevent a corrupt party from injecting transactions after it has already observed other transactions, nor a corrupt leader from reordering transactions after reading them, violating both Blind and Time-Based Order-Fairness.

2.3 Threshold Methods

Beside standard cryptographic primitives, like a symmetric encryption scheme or a cryptographic hash function $H()$, we employ primitives specifically designed for hiding and opening transactions.

Secret Sharing. For positive integers k, n, where $k \leq n$, a *(k,n)-threshold scheme* is a technique where a secret is distributed among n parties , such that it can be reconstructed by any k parties, while any group of $k-1$ parties cannot learn the secret. The secret S is selected by a trusted third party, called *dealer*, who splits it into n individual *shares* by calling: $s_1, ..., s_n \leftarrow \texttt{SS.Split(S)}$. After at least k parties exchange shares, they can combine them and recover the secret by calling $S \leftarrow \texttt{SS.Combine(}s'_1, ..., s'_k\texttt{)}$.

Shamir's Secret Sharing (SSS) [38] is one of the first and most widely used *(k,n)*-threshold schemes. In SSS, $\texttt{Split(}S\texttt{)}$ chooses a polynomial f of degree $k-1$ that hides S at the origin point, and generates n points on the polynomial. Then, any k points allow to interpolate the polynomial and compute its value at $f(0)$.

Threshold Encryption. A *(k,n)-threshold encryption* is a scheme where messages encrypted under a single public encryption key pk can be decrypted by a private decryption key, shared among n parties – each with its secret key sk_i, where any group of k parties can decrypt a message. The message m is encrypted with an encryption algorithm: $c \leftarrow \texttt{TE.Enc(}pk, m\texttt{)}$.

We consider schemes where each party, upon receiving a ciphertext c, computes its own *decryption share* by invoking: $ds_i \leftarrow$ `TE.ShareGen(`sk_i, c`)`; and any set of k unique decryption shares is enough to recover the plaintext, which we denote $m \leftarrow$`TE.Dec(`$c, ds'_1, ..., ds'_k$`)`.

Moreover, some threshold encryption schemes, like shown by Shoup and Gennaro [39], allow parties to verify that a given decryption share corresponds to the ciphertext, which we denote by the `TE.Verify(`c, ds_i`)` function. Note that, without such a verification, it is trivial for a malicious party, providing an incorrect share, to cause decryption failures.

3 The commit-reveal Framework

The core of blind fairness-ordering is a standard k-of-n secret sharing approach. For each transaction `tx`, a user ("Alice") picks a secret symmetric key `tx-key`, and sends `tx` encrypted with it to validators. Secret sharing allows Alice to share `tx-key` with validators such that $F+1$ shares are required to reconstruct `tx-key`, and no set of F bad validators can open it before it is committed (blindly) to the total order. Honest validators wait to commit to a blind order of transactions first, and only later open them. At the same time, before committing a transaction to the total ordering, validators ensure that they can open it.

More formally, a sharing protocol has two abstract functionalities, Disperse(tx) and Retrieve(tx). Disperse() allows a dealer to disseminate shares of the secret `tx-key` to validators, with the following guarantees:

Hiding: A set of F bad validators cannot reconstruct `tx-key`.

Binding: After Disperse(tx-key) completes successfully at an honest validator, any set of honest $F+1$ validators doing Retrieve(tx-key) can reconstruct `tx-key` generating a unique outcome.

Validity: If the dealer is honest, then the outcome from Retrieve(tx-key) by any $F+1$ honest validators is `tx-key`.

It is worth noting that Disperse()/Retrieve() does not require certain properties which some use-cases in the literature may need, but not the framework here:

- When Disperse() completes, it does not guarantee that Retrieve() can reconstruct an output value that a dealer commits to.
- It doesn't require that individual shares can be recovered.
- It doesn't require being able to use `tx-key` to derive other values while keeping `tx-key` itself secret.

We proceed to present four approaches for implementing Disperse()/Retrieve(). All of them are compatible with the DAG-based consensus protocol in the next section, though the fastest and simplest one, which we recommend using, is AVID-M.

Approach-1: Threshold Cryptography

It is straight-forward to implement Disperse/Retrieve using threshold encryption, such that the public encryption `TE.Enc()` is known to users and the private decryption `TE.Dec()` is shared (at setup time) among validators. To Disperse(tx-key), the user attaches `TE.Enc(tx-key)`, the transaction key encrypted with the global threshold key. Once a `tx` is committed to the total-ordering, Retrieve(tx-key) is implemented by each validator generating its decryption share via `TE.ShareGen()`. Validators can try applying `TE.Dec()` over $F+1$ valid decryption shares to decrypt `tx`.

Binding stems from two properties. First, a threshold of honest validators can always succeed in generating $F+1$ valid decryption shares. Second, as indicated in section 2, some threshold cryptography schemes allow verifying that a validator is contributing a correct decryption share, hence by retrieving $F+1$ valid threshold shares, a unique outcome is guaranteed.

Unfortunately, this method is computationally somewhat heavy: Disperse() takes about $300\mu s$ to encrypt using TDH2 on standard hardware, and Retrieve() takes about $3500\mu s$ for in a 6-out-of-16 scheme (see section 7).

Approach-2: VSS

Another way for users to Disperse(tx-key) is Shamir's secret sharing (SSS) scheme. To Disperse(tx-key), a user employs `SS.Split(tx-key)` to send individual shares of `tx-key` to each validator. A set of $F+1$ share holders reveal their shares during Retrieve() and combine shares via `SS.Combine()` to reconstruct `tx-key`.

Combining shares is three orders of magnitude faster than threshold crypto and takes a few microseconds in today's computing environment (see section 7). However, after Disperse() completes with $N-F$ validators, "vanilla" `SS.Split()/SS.Combine()` does not guarantee Binding: first, not every set of $F+1$ honest validators may hold shares from the Disperse() phase. Second, retrieving shares from different sets of $F+1$ validators may result in different outcome from `SS.Combine()`.

VSS schemes support Binding through a share-verification functionality `VSS.Verification(share, commit value)`. Verification allows validators to check that shares are consistent with some committed value S', such that any set of $F+1$ shares applied to `SS.Combine()` results in output S'. When a validator receives a share, it should verify the share before acknowledging it. Disperse() completes when there are $N-F$ acknowledgements of verifiable shares, certifying that valid shares are held by $F+1$ honest validators and guaranteeing a unique reconstruction by any subset of $F+1$. It is possible to add a share-recovery functionality to allow a validator to obtain its individual share prior to Retrieve().

However, despite the vast progress in VSS schemes (see subsection 8.1), share recovery requires linear communication. Additionally, implementing VSS requires non-trivial cryptography.

AVID-M

VSS provides stronger guarantees than necessary to satisfy the Binding requirement of commit-reveal. Specifically, Disperse() only needs to guarantee a unique outcome from Retrieve(), not success. Borrowing a technique called AVID-M, which was introduced in DispersedLedger [40] for reliable information dispersal, validators can completely avoid verifying shares during Disperse(). Instead, they verify during Retrieve() that the (entire) sharing would have a unique outcome, which is very cheap to do. This works as follows.

To Disperse(tx), a user employs `SS.Split(tx-key)` to send individual shares to validators. This operation requires a deterministic way of splitting secret shares which, in particular, can be based on validator identifiers (e.g., $s_i = P(id_i)$, where P is the polynomial hiding `tx-key`). Additionally, the user combines all shares in a Merkle-tree, certifies the root, and sends with each share a proof of membership, i.e., a Merkle tree path to the root.

When a validator receives a share, it should verify the Merkle tree proof against the certified root before acknowledging the share. Disperse() completes when there are $N-F$ acknowledgements, guaranteeing that untampered shares are held by $F+1$ honest validators.

During Retrieve(), $F+1$ validators reveal individual shares, attaching the Merkle tree path to prove shares have not been tampered with. Note that $F+1$ shares can be validated by checking only one signature on the root. Validators use `SS.Combine()` to try to reconstruct `tx-key`.

Then they *post-verify* that every subset of $F+1$ (untampered) shares sent to validators would have the same outcome. To do this, a validator does not need to wait for missing shares nor try combinations of $F+1$ shares. Moreover, it does not need to communicate with others. A validator simply generates missing shares and re-encodes the Merkle tree. Then, it compares the re-generated Merkle tree with the root certified by the sender. If the comparison fails, the dealer was faulty and the validator rejects `tx`. The unique reconstruction is guaranteed because the post-verification outcome – succeed or fail – becomes fixed upon Disperse completion. Each validator arrives at the same outcome no matter which subset of $F+1$ (untampered) shares are input.

AVID-M takes roughly $50\mu s$ to `SS.split()` and generate Merkle tree commitments, and roughly $50\mu s$ for Retrieve(), including post-verification, in a 6-out-of-16 scheme.

Hybrid

Even after Disperse() has completed, both VSS and AVID-M may need to interact with a specific set of $F+1$ honest validators during Retrieve(). This implies that the latency of Retrieve() is impacted by this specific set of $F+1$ share holders, and not the speed of the fastest $F+1$ honest validators. Employing threshold cryptography removes this dependency but incurs a costly computation.

A Hybrid approach combines the benefits of threshold cryptography with AVID-M. In Hybrid, Retrieve() works with (fast) secret-sharing during steady-state, and falls back to threshold cryptography, avoiding waiting for specific $F+1$ validators, during a period of network instability.

Disperse(tx-key) is implemented in two parts. First, a user applies AVID-M to send validators individual shares and proofs. Second, as a fallback, it sends validators `TE.Enc(tx-key)`. Once `tx` is committed to the total ordering, Retrieve(tx-key) has a fast track and a slow track. In the fast track, every validator that holds an AVID-M share reveals it. A validator that doesn't hold an AVID-M share for `tx-key` reveals a threshold key decryption share. In the slow track, validators may give up on waiting for AVID-M shares and reveal threshold key decryption shares, even if they already revealed AVID-M shares.

Post-verification after $F+1$ valid shares of either kind are revealed checks that **both** the AVID-M and threshold encryption were shared correctly and would have the same outcome. More specifically, a validator both re-encrypts `tx-key` and re-encodes the AVID-M Merkle tree after reconstruction. It compares both outcomes with the sender's. If the comparison fails, the dealer was faulty and the validator rejects `tx`. Binding holds because the post-verification outcome – succeed or fail – becomes fixed upon Disperse() completion. Each validator arrives at the same outcome no matter which scheme and what subset of $F+1$ (untampered) shares are retrieved, thus ensuring Binding.

4 DAG transport

In a DAG-based BFT protocol, validators store messages delivered via reliable and causally ordered broadcast in a local graph. A message inserted into the local DAG has the following guarantees: *Reliability*: there are copies of the message stored on sufficiently many participants, such that eventually, all honest validators can download it. *Non-equivocation*:

messages by each validator are numbered. If a validator delivers some message as the k'th from a particular sender, then the message is authenticated by its sender and other validators deliver the same message as the sender k'th message. *Causal Ordering*: the message carries explicit references to messages which the sender has previously delivered (including its own). Predecessors are delivered locally before the message.

Note that the DAGs at different validators may be slightly different at any moment in time. This is inevitable in a distributed system due to message scheduling. However, a DAG-based Consensus protocol allows each participant to interpret its local DAG, reaching an autonomous conclusion that forms total ordering. Reliability, Non-equivocation and Causal Ordering make the design of such protocols extremely simple as we shall see below.

DAG API. A DAG transport exposes two basic API's, `broadcast()` and `deliver()`.

Broadcast. `broadcast()` is an asynchronous API for a validator to present payload for the DAG transport to be transmitted to all other validators. The main use of DAG messages in a DAG-based Consensus protocol is to pack meta-information on transactions into a block and present the block for broadcast. The DAG transport adds references to previously delivered messages, and broadcasts a message carrying the block and the causal references to all validators.

Deliver. When another validator receives the message carrying the block, it checks whether it needs to retrieve a copy of any of the transactions in the block and in causally preceding messages. Once it obtains a proof-of-availability of all transactions in the block and its causal past, it can acknowledge it. A validator's upcall `deliver(m)` is triggered when sufficiently many acknowledgments for it are gathered, guaranteeing that the message itself, the transactions it refers to, and its entire causal past maintain Reliability, Non-equivocation and Causal Ordering.
It is worth noting that in the protocol discussed in this paper, the Consensus protocol injects meta-information (e.g., complaints), DAG broadcast never stalls or waits for input from it.

Implementing a DAG. There are various ways to implement reliable, non-equivocating and causally-ordered broadcast among $N = 3F + 1$ *validators*, at most F of which are presumed Byzantine faulty and the rest are honest.

Lifetime of a message. A validator packs transaction information and meta-information into a message, adds references to previously delivered messages (including the sender's own preceding messages), and broadcasts the message to all validators.

Echoing. The key mechanism for reliability and non-equivocation is for validators to echo a digest of the first message they receive from a sender with a particular index. When $2F+1$ echoes are collected, the message can be delivered. There are two ways to echo, one is all-to-all broadcast over authenticated point-to-point channels a la Bracha Broadcast [9]; the other is converge-cast with cryptographic signatures a la Rampart [37] and Cachin et al. [10]. In either case, echoing can be streamlined so the amortized per-message communication is linear, which is anyway the minimum necessary to spread the message.

Layering. Transports are often constructed in a layer-by-layer regime. In this regime, each sender is allowed one message per layer, and a message may refer only to messages in the layer preceding it. Layering is done so as to regulate transmissions and saturate network capacity, and has been demonstrated to be highly effective by various projects [13, 19, 14]. We reiterate that Fino does not require a layered structure.

5 Fino

Fino incorporates MEV protection into a BFT protocol for the partial synchrony model, riding on a DAG transport. BFT validators periodically pack pending encrypted transactions into a batch and use the DAG transport to broadcast them. The key insight for operating commit-reveal on a DAG is that each view must wait until Retrieve() completes on transactions `tx` of the previous view.

5.1 The Protocol

Views. The protocol operates in a view-by-view manner. Each view is numbered, as in `view(r)`, and has a designated leader known to everyone.

View change. A validator enters view(r+1) when two conditions are met, viewA and viewB: viewA is satisfied when the local DAG of a validator contains $F + 1$ valid votes on `proposal(r)` or $2F + 1$ valid `complaint(r)` on `view(r)`. viewB is satisfied when every committed transaction `tx` in the local DAG has $F+1$ valid shares revealed, hence Retrieve(tx) can be completed. Note that viewB prevents BFT validators that do not reveal (correct) shares from enabling the protocol to make progress without opening committed transactions.

Proposing. When a leader enters a new view(r), it broadcasts `proposal(r)`.[1] Implicitly, `proposal(r)` suggests to commit to the global ordering of transactions all the messages in the causal history of the proposal. A leader's `proposal(r)` is *valid* if it is well-formatted and is justified in entering `view(r)`.

Voting. When a validator sees a valid leader proposal, it broadcasts `vote(r)`. A validator's `vote(r)` is *valid* if it follows a valid `proposal(r)`.

Committing. Whenever a leader's `proposal(r)` has $F+1$ valid votes in the local DAG, the proposal and its causal history become *committed*. The commit order induced by a commit decision is described below.

Share revealing. When a validator observes that a transaction `tx` becomes committed, it starts Retrieve(tx) and calls `broadcast()` to present its share of the decryption key `tx-key`.

Complaining. If a validator gives up waiting for a commit to happen in `view(r)`, it broadcasts `complaint(r)`. Note, a `vote(r)` by a validator that causally follows a `complaint(r)` by the validator, if exists, is **not** interpreted as a valid vote.

Ordering Commits. When a validator observes that a leader's `proposal(r)` becomes committed, it orders newly committed transactions as follows:

1. Let `r'` be the highest view `r' < r` for which `proposal(r')` is in the causal history of `proposal(r)`. `proposal(r')` is recursively ordered.

[1] Recall, broadcast() merely presents payload to be transmitted as scheduled by the DAG transport, e.g., piggybacked on messages carrying other transaction info.

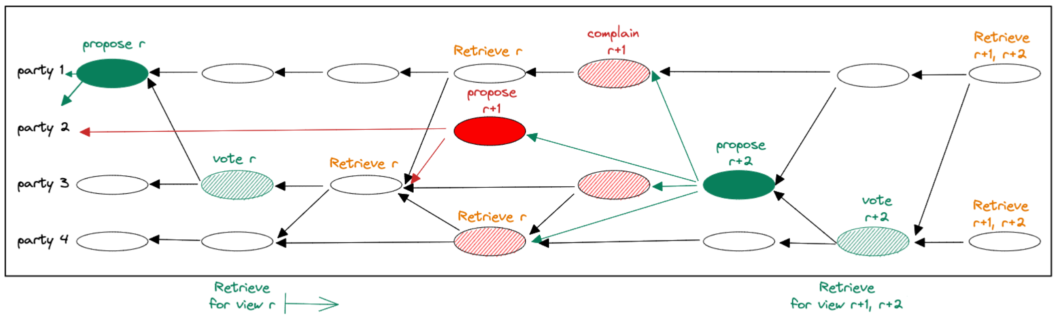


Figure 1 A commit of `proposal(r)` is followed by share retrieval. A commit of `proposal(r+2)` causes an indirect commit of `proposal(r+1)`, followed by share revealing of both.

2. The remaining causal predecessors of `proposal(r)` which have not yet been ordered are appended to the committed sequence (within this batch, ordering can be done using any deterministic rule to linearize the partial ordering into a total ordering.)

Opening Transactions. When a validator observes that a leader's `proposal(r)` becomes committed, it decrypts every committed transaction `tx` in its causal past that hasn't been decrypted already. That is, let `c` be the highest view `c < r` for which `proposal(r)` causally follows $F + 1$ valid votes. Transactions in the causal past of `proposal(c)` are opened as follows:

1. Let `r'` be the highest view `r' < c` for which `proposal(r')` is in the causal history of `proposal(c)`. `proposal(r')` is recursively opened.
2. The remaining (committed) transactions `tx` in `proposal(c)`'s causal past are opened using Retrieve(tx) to either produce a key `tx-key` that decrypts `tx` or to reject `tx`.

Figure 1 illustrates a couple of Fino scenarios.

Happy-path scenario. In the first view (view(r)), `proposal(r)` becomes committed. The commit sets an ordering for transactions in the causal past of `proposal(r)`, enabling retrieval of shares for transactions from `proposal(r)`.

Scenario with a slow leader. A slightly more complex scenario occurs when a view expires because validators do not observe a leader's proposal becoming committed and they broadcast complaints. Figure 1 depicts this happening in view(r+1). Entering view(r+2) is enabled by $2F + 1$ complaints about view(r+1). When `proposal(r+1)` itself becomes committed, it indirectly commits `proposal(r+1)` as well. Thereafter, validators reveal shares for all pending committed transactions, namely, those in both `proposal(r+1)` and `proposal(r+2)`.

6 Analysis

Fino is minimally integrated with a DAG transport. BFT commit-reveal logic is embedded into the DAG structure simply by broadcasting (more precisely, injecting payloads into broadcasts) in the form of protocol proposals/votes/complaints and revealing shares. Importantly, at no time is DAG broadcast slowed down by the Fino protocol. The reliability and causality properties of the DAG transport make arguing about safety and liveness relatively easy.

6.1 Safety

▶ **Lemma 1.** *Assume a* `proposal(r)` *ever becomes committed. Denote by r' the minimal view, where $r' > r$, such that* `proposal(r')` *ever becomes committed. Then for every $r \leq q < r'$,* `proposal(q+1)` *causally follows* `proposal(r)`*.*

Proof (Sketch). Since `proposal(r)` becomes committed, by definition $F + 1$ parties sent votes for it. There are two possibilities for a leader's `proposal(q+1)` to be valid. The first is `proposal(q+1)` may reference $F + 1$ valid `vote(q)` messages. This case occurs only for $q == r$ by the lemma assumption that $r' \geq q + 1$ is the minimal view that ever becomes committed. The second possibility can occur for all $r \leq q < r'$, namely, `proposal(q+1)` references $2F + 1$ valid `complaint(q)` messages.

In the first case, valid `vote(r)` causally follows `proposal(r)`, and a fortiori `proposal(r+1)` causally follows `proposal(r)`.

In the second case, one of $2F + 1$ `complaint(q)` messages is sent by a party who sent a valid `vote(r)`. By definition, `vote(r)` must precede `complaint(q)`, otherwise it is not considered a (valid) vote. Hence, `proposal(q+1)` causally follows `complaint(q)` which follows `proposal(r)`. ◀

▶ **Lemma 2.** *If ever a* `proposal(r)` *becomes committed, then every valid* `proposal(q)`*, where* `q > r`*, causally follows* `proposal(r)`*.*

Proof (Sketch). The proof follows by a simple induction on Lemma 1. ◀

▶ **Lemma 3.** *If an honest party commits* `proposal(r)`*, and another honest party commits* `proposal(r')`*, where $r' > r$, then the sequence of transactions committed by* `proposal(r)` *is a prefix of the sequence committed by* `proposal(r')`*.*

Proof (Sketch). When `proposal(r')` becomes committed, the commit ordering rule is recursively applied to valid proposals in its causal past. By Lemma 2, the (committed) `proposal(r)` is a causal predecessor of every valid `proposal(s)`, for $r < s \leq r'$, and eventually the recursion gets to it. ◀

▶ **Lemma 4.** *If ever a* `proposal(r)` *becomes committed, then every committed transaction proposed by a valid* `proposal(r')`*, where* `r' < r`*, can be uniquely decrypted.*

Proof (Sketch). When `proposal(r)` becomes committed, then by Lemma 3, all the transactions from previously committed proposals are part of the committed transaction history. Moreover, committing `proposal(r)` implies the view change (from `r-1`) which occurs only if the condition viewB (see section 5) is met, i.e., every committed transaction `tx` in the local DAG can complete `Retrieve(tx)`, hence can be uniquely decrypted (or rejected). ◀

6.2 Liveness

Liveness is guaranteed after GST, i.e., after communication has become synchronous with a known Δ upper bound on transmission delays.

After GST, views are semi-synchronized through the DAG. In particular, suppose that after GST, every broadcast by an honest party arrives at all honest parties within DD. Once a `view(r)` with an honest leader is entered by the first honest party, within DD all the messages seen by one party are delivered by both the leader and all other honest parties. Hence, within DD, all honest parties enter `view(r)` as well. Within two additional DAG latencies, $2 \cdot DD$, the `view(r)` proposal and votes from all honest parties are spread to everyone.

Assuming view timers are set to be at least $3 \cdot DD$, once `view(r)` is entered, a future view will not interrupt a commit. In order to start a future view, its leader must collect either $F + 1$ `vote(r)` messages, hence commit `proposal(r)`; or $2F + 1$ `complaint(r)` expiration messages, which is impossible as argued above.

6.3 Communication complexity

Protocols for the partial synchrony model have unbounded worst case by nature, hence, we concentrate on the costs incurred during steady state when a leader is honest and communication with it is synchronous:

DAG message cost. In order for DAG messages to be delivered reliably, it must implement reliable broadcast. This incurs either a quadratic number of messages carried over authenticated channels, or a quadratic number of signature verifications, per broadcast. In either case, the quadratic cost may be amortized by pipelining, driving it in practice to (almost) linear per message.

Commit message cost. Fino sends $F + 1$ broadcast messages, a proposal and votes, per decision. A decision commits the causal history of the proposal, consisting of (at least) a linear number of messages. Moreover, each message may carry multiple transactions in its payload. As a result, in practice the commit cost is amortized over many transactions.

6.4 Latency

Commit latency. The commit latency in terms of DAG messages is 2, one proposal followed by votes.

Opening latency. During periods of stability, there are no complaints about honest leaders by any honest party. If `tx` is proposed by an honest leader in `view(r)`, it will receive $F + 1$ votes and become committed within two DAG latencies. Within one more DAG latency, every honest party will post a message containing a share for `tx`. Thereafter, whenever $F + 1$ are available, everyone will be able to complete Retrieve(tx) with a unique outcome.

6.5 Faulty User

One drawback of the proposed protocol is that it assumes a well-connected and honest user (dealer), able to entrust $F + 1$ honest validators with their shares. If the dealer is malicious or unable to broadcast shares it can lead to the situation where the transport layer buffers a transaction indefinitely without being able to `deliver()` it. The Hybrid approach we presented in section 3 solves this issue by using threshold encryption as a fallback mechanisms (i.e., whenever enough shares cannot be received, parties would proceed with threshold decryption). However, the hybrid construction suffers the slowness of threshold encryption to verify a unique outcome even in faultless executions.

7 Evaluation (prelim)

We implemented the proposed variants of the Fino Disperse and Retrieve functionalities and investigated the computational overhead that these schemes introduce. For secret sharing we implemented SSS [38], while for threshold encryption we implemented the TDH2 scheme [39].

Table 1 Comparison of the implementations' performance.

Scheme	`Disperse`	`ShareGen`	`ShareVerify`	`Reconstruct`
TDH2-based	311.6μs	434.8μs	492.5μs	763.9μs
SSS-AVID-M	52.7μs	N/A	2.7μs	52.5μs
Hybrid opt.	360.3μs	N/A	2.7μs	361.5μs
Hybrid pess.	360.3μs	434.8μs	492.5μs	828.9μs

We selected the schemes with the most efficient cryptographic primitives we had access to, i.e., the secret sharing scheme uses the Ed25519 curve [7], while TDH2 uses ristretto255 [15] as the underlying prime-order group. Performance for both schemes is presented in a setting where 6 shares out of 16 are required to recover the plaintext.

The results presented in Table 1 are obtained on an Apple M1 Pro. `Disperse` refers to the overhead on the client-side while `ShareGen` is the operation of deriving a decryption share from the TDH2 ciphertext (it is absent in SSS and the optimistic path of the hybrid scheme). In TDH2, `ShareVerify` verifies if a decryption share matches the ciphertext, while in SSS-based Fino it only checks whether a share belongs to the tree aggregated by the signed Merkle root attached by the client. `Reconstruct` recovers the plaintext from the ciphertext and the number of shares and verifies the outcome integrity (i.e., AVID-M and threshold re-encryption verification, where applicable).

As demonstrated by these micro-measurements, the SSS-AVID-M scheme is the most efficient: in our blind ordering scenario using threshold encryption, each party processing a TDH2 ciphertext would call `ShareGen` once to derive its decryption share, `ShareVerify` *k-1* times to verify the threshold number of received shares, and `Reconstruct` once to obtain the plaintext. Assuming *k=6*, the total computational overhead for a single transaction to be recovered would take around 3.7ms CPU time. With secret-sharing based Fino, the party would also call `ShareVerify` *k-1* times and `Reconstruct` once, which requires only 66μs CPU time. The hybrid approach introduces an interesting trade-off between its properties and performance. In the optimistic path, it requires around 378μs CPU time per transaction, while its fallback introduces negligible overhead to the plain threshold-encryption scheme.

We emphasize that these are micro-benchmarks of the blinding/unblinding of transactions; in the future, we plan to complete a performance evaluation of the entire DAG-based blind-ordering protocol.

8 Related Work and Discussion

Despite the MEV problem being relatively new, there already exist consensus-related systems aiming at solving it. For instance, Flash Freezing Flash Boys (F3B) [42] is a commit-and-reveal architecture, where clients send TDH2-encrypted ciphertexts to be ordered by Consensus, and afterward to be decrypted via shares released by a dedicated secret-management committee. Other closely related techniques and schemes helpful in mitigating MEV are discussed below.

8.1 Verifiable Secret Sharing (VSS)

Although we ultimately end up forgoing verifiability/recoverability of shares, VSS is used in many settings like ours. The overall communication complexity incurred in VSS on the dealer sharing a secret and on a party recovering a share has dramatically improved in recent years. VSS can be implemented inside the asynchronous echo broadcast protocol in $O(n^3)$

communication complexity using Pederson's original two-dimensional polynomial scheme Non-Interactive Polynomial Commitments [34]. Kate et al. [22] introduce a VSS scheme with $O(n^2)$ communication complexity, utilized for asynchronous VSS by Backes et al. [4]. Basu et al. [5] propose a scheme with linear $O(n)$ communication complexity.

A related notion is robust secret sharing (RSS) introduced by Krawczyk [27] and later revised by Bellare and Rogaway [6]. RSS allows recoverability in case of incorrect (not just missing) shares, however, these schemes assume an honest dealer. Duan et al. [17] propose ARSS extending RSS to the asynchronous setting.

8.2 Time-Based Order-Fairness

Blind Order-Fairness is achieved by deterministically ordering encrypted transactions and then, after the order is final, decrypting them. The deterministic order can be enhanced by sophisticated ordering logic present in other protocols. In particular, Fino can be extended to provide Time-Based Fairness additionally ensuring that the received transactions are not only unreadable by parties, but also their relative order cannot be influenced by malicious parties (the transaction order would be defined by the time of its ciphertext arrival).

For instance, Pompē [43] proposes a property called *Ordering Linearizability*: *"if all correct parties timestamp transactions tx, tx' such that tx' has a lower timestamp than tx by everyone, then tx' is ordered before tx."* It implements the property based on an observation that if parties exchange transactions associated with their receiving timestamps, then for each transaction its median timestamp, computed out of 2F+1 timestamps collected, is between the minimum and maximum timestamps of honest parties. Fino can be easily extended by the Linearizability property offered by Pompē and the final protocol is similar to the Fino with Blind Order-Fairness (see above) with only one modification. Namely, every time a new batch of transactions becomes committed, parties independently sort transactions by their aggregate (median) timestamps.

More generally, Fino can easily incorporate other Time-based Fairness ordering logic. Note that in Fino, the ordering of transactions is determined on encrypted transactions, but time ordering information should be open. The share revealing, share collection, and unique decryption following a committed ordering are the same as presented previously. The final protocol offers much stronger properties since it not only hides payloads of unordered transactions from parties, but also prevents parties from reordering received transactions.

One form of Time-based Order-Fairness introduced in Aequitas [26] is *Batch-Order Fairness*: *"if sufficiently many (at least ½ of the) parties receive a transaction tx before another transaction tx', then no honest party can deliver tx in a block after tx', "* and a protocol achieving it. Other forms of Time-Based Order-Fairness which may be used in Fino include "Wendy Grows Up" [28], which introduced *Timed Relative Fairness*, *"if there is a time t such that all honest parties saw (according to their local clock) tx before t and tx' after t , then tx must be scheduled before tx', "* and "Quick Order Fairness" [11], which defined *Differential-Order Fairness*, *"when the number of correct parties that broadcast tx before tx' exceeds the number that broadcast tx' before tx by more than $2F + \kappa$, for some $\kappa \geq 0$, then the protocol must not deliver tx' before tx (but they may be delivered together)."*

Themis [25] is a protocol realizing *Batch-Order Fairness*, where parties do not rely on timestamps (as in Pompē) but only on their relative transaction orders reported. Themis can also be integrated with Fino, however, to make it compatible this design requires some modifications to Fin's underlying DAG protocol. More concretely, Themis assumes that the fraction of bad parties cannot be one quarter, i.e., F out of 4F+1. A leader makes a proposal based on 3F+1 out of 4F+1 transaction orderings (each reported by a distinct party). Therefore, we would need to modify the DAG transport so that parties reference 3F+1 preceding messages (rather than 2F+1).

8.3 DAG-based BFT

There are many known BFT solutions for partial synchrony, and more specifically, several recent solutions that ride on a DAG [14, 20, 24]. When constructing Fino, we wanted to build MEV protection into a simple DAG-based BFT solution, described below. Notwithstanding, we remark that Fino's MEV protection can possibly be incorporated into other DAG-riding BFT solutions.

We borrowed a subprotocol from Bullshark [20] that deals with partial synchrony, modifying it so that DAG transmissions would never stall waiting for BFT protocol steps or timers. Fino is different from the borrowed Bullshark component in that BFT protocol steps (e.g., view changes, proposals, votes, and complaints) are injected into the DAG at any time, independent of DAG layers. It uses `broadcast()` (defined in section 2) to present these steps as payloads to the DAG transport, completely asynchronously, while normal DAG transmissions continue. A hallmark of the DAG-riding approach, which Fino preserves while adding Blind Order-Fairness, is zero message overhead.

Narwhal [14] is a recent DAG transport that has a layer-by-layer structure, each layer having at most one message per sender and referring to 2F+1 messages in the preceding layer. A similarly layered DAG construction appears earlier in Aleph [19], but it does not underscore the separation of transaction dissemination from DAG messages. Narwhal-HS is a BFT Consensus protocol within [14] based on HotStuff [41] for the partial synchrony model, in which Narwhal is used as a "mempool". In order to drive Consensus decisions, Narwhal-HS adds messages outside Narwhal, using the DAG only for spreading transactions.

DAG-Rider [23] and Tusk [14] build randomized BFT Consensus for the asynchronous model riding on Narwhal, These protocols are zero message overhead over the DAG, not exchanging any messages outside the Narwhal protocol. Both are structured with purpose-built DAG layers grouped into "waves" of 4 (2) layers each. Narwal waits for the Consensus protocol to inject input value every wave, though in practice, this does not delay the DAG materially.

Bullshark [20] builds BFT Consensus riding on Narwhal for the partial synchrony model. It is designed with 8-layer waves driving commit, each layer purpose-built to serve a different step in the protocol. Bullshark is a "zero message overhead" protocol over the DAG, however, due to a rigid wave-by-wave structure, the DAG is modified to wait for Bullshark timers/steps to insert transactions into the DAG. In particular, if leader(s) of a wave are faulty or slow, some DAG layers wait to fill until consensus timers expire.

More generally, since the eighties, causally ordered reliable broadcast has been utilized in forming distributed Consensus protocols, e.g., [8, 35, 30, 32, 3, 16].

The notion of the secure, causal, reliable broadcast was introduced by Reiter and Birman [37], and later refined by Cachin [10] and Duan et al. [17]. This primitive was utilized in a variety of BFT replicated systems, but not necessarily in the form of zero message overhead protocols riding on a DAG. Several of these pre blockchain-era BFT protocols are DAG based, notably Total [32] and ToTo [16], both of which are BFT solutions for the asynchronous model.

9 Conclusions

In this paper, we show how DAG-based BFT protocols can be enhanced to mitigate MEV – arguably one of the main threats to the success of cryptocurrencies. We investigate the design space of achieving Blind Order-Fairness, and we propose an approach that focuses on practicality. We present preliminary correctness and performance results which indicate that our scheme is secure and efficient. In the future, we plan to extend our analysis and experiments, and investigate other cryptographic tools which can improve our scheme.

References

1 Mev-explore v1. `https://explore.flashbots.net/`, 2022. Accessed: 2022-07-18.

2 Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE TDSC*, 2010.

3 Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. *Transis: A communication sub-system for high availability.* Hebrew University of Jerusalem. Leibniz Center for Research in Computer . . . , 1991.

4 Michael Backes, Amit Datta, and Aniket Kate. Asynchronous computational vss with reduced communication complexity. In *CT-RSA*. Springer, 2013.

5 Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K Reiter, and Emin Gün Sirer. Efficient verifiable secret sharing with share recovery in bft protocols. In *ACM CCS*, 2019.

6 Mihir Bellare and Phillip Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *ACM CCS*, 2007.

7 Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2012.

8 Ken Birman and Thomas Joseph. Exploiting virtual synchrony in distributed systems. In *ACM SOSP*, 1987.

9 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 1987.

10 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*. Springer, 2001.

11 Christian Cachin, Jovana Mićić, and Nathalie Steinhauer. Quick order fairness. *arXiv preprint*, 2021. `arXiv:2112.06615`.

12 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE SP*. IEEE, 2020.

13 George Danezis and David Hrycyszyn. Blockmania: from block dags to consensus. *arXiv preprint*, 2018. `arXiv:1809.01620`.

14 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *EuroSys*, 2022.

15 Henry de Valence, Jack Grigg, George Tankersley, Filippo Valsorda, and Isis Lovecruft. The ristretto255 group. IETF CFRG Internet Draft, 2020.

16 Danny Dolev, Shlomo Kramer, and Dalia Malki. Early delivery totally ordered multicast in asynchronous environments. In *FTCS*. IEEE, 1993.

17 Sisi Duan, Michael K Reiter, and Haibin Zhang. Secure causal atomic broadcast, revisited. In *IEEE/IFIP DSN*. IEEE, 2017.

18 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 1988.

19 Adam Gągol and Michał Świętek. Aleph: A leaderless, asynchronous, byzantine fault tolerant consensus protocol. *arXiv preprint*, 2018. `arXiv:1810.05256`.

20 Neil Giridharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Bullshark: Dag bft protocols made practical. *arXiv preprint*, 2022. `arXiv:2201.05677`.

21 Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint*, 2022. `arXiv:2203.11520`.

22 Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*. Springer, 2010.

23 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *ACM PODC*, 2021.

24 Idit Keidar, Oded Naor, and Ehud Shapiro. Cordial miners: A family of simple, efficient and self-contained consensus protocols for every eventuality. *arXiv preprint*, 2022. `arXiv:2205.09174`.

25 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *Cryptology ePrint Archive*, 2021.

26 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*. Springer, 2020.

27 Hugo Krawczyk. Secret sharing made short. In *CRYPTO*. Springer, 1993.

28 Klaus Kursawe. Wendy grows up: More order fairness. In *Financial Crypto*. Springer, 2021.

29 Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Fairledger: A fair blockchain protocol for financial institutions. *arXiv preprint*, 2019. arXiv:1906.03819.

30 Peter M Melliar-Smith, Louise E. Moser, and Vivek Agrawala. Broadcast protocols for distributed systems. *IEEE TPDS*, 1990.

31 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *ACM CCS*, 2016.

32 Louise E Moser and Peter M Melliar-Smith. Byzantine-resistant total ordering algorithms. *Information and Computation*, 1999.

33 Alexandre Obadia, Alejo Salles, Lakshman Sankar, Tarun Chitra, Vaibhav Chellani, and Philip Daian. Unity is strength: A formalization of cross-domain maximal extractable value. *arXiv preprint*, 2021. arXiv:2112.01472.

34 Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*. Springer, 1991.

35 Larry L Peterson, Nick C Buchholz, and Richard D Schlichting. Preserving and using context information in interprocess communication. *ACM TOCS*, 1989.

36 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint*, 2021. arXiv:2101.05511.

37 Michael K Reiter and Kenneth P Birman. How to securely replicate services. *ACM TOPLAS*, 1994.

38 Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.

39 Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *EUROCRYPT*. Springer, 1998.

40 Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-throughput byzantine consensus on variable bandwidth networks. In *USENIX NSDI*, 2022.

41 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *ACM PODC*, 2019.

42 Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. Flash freezing flash boys: Countering blockchain front-running. In *DINPS*, 2022.

43 Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *USENIX OSDI*, 2020.

Commit-Reveal Schemes Against Front-Running Attacks

Andrea Canidio
IMT School for Advanced Studies, Lucca, Italy
CoW Protocol, Paris, France

Vincent Danos
CNRS, Paris, France
École Normale Supérieure, Paris, France

Abstract

We provide a game-theoretic analysis of the problem of front-running attacks. We use it to study a simple commit-reveal protocol and discuss its properties. This protocol has costs because it requires two messages and imposes a delay. However, we show that it prevents the most severe front-running attacks ("bad MEV") while preserving legitimate competition between users, guaranteeing that the earliest transaction in a block belongs to the honest user who values it the most ("good MEV").

2012 ACM Subject Classification Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Front running, Game theory, MEV, Transactions reordering, commit-reveal

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.7

Category Extended Abstract

Related Version *Full Version*: `https://arxiv.org/abs/2301.13785`

Funding We gratefully acknowledge the financial support of the Ethereum Foundation (grant FY22-0840).

Acknowledgements We are grateful to Agostino Capponi, Jiasun Li, Christof Ferreira Torres, Arthur Gervais, Ari Juels, and the participants to UBRI Connect 2022, Tokenomics 2022 for their comments and suggestions.

1 Introduction

On the Ethereum network, each validator decides how to order pending transactions to form the next block, hence determining the order in which these transactions are executed. As a consequence, users often compete with each other to have their transactions included earlier in a block, either by paying transaction fees or by making side payments directly to validators.[1] This form of competition can be beneficial because it ensures that a scarce resource (i.e., having a transaction included earlier in the block) is allocated to the user who values it the most.[2] But at the same time, it opens the possibility of front-running attacks: because pending transactions are public, a malicious user can observe a victim's incoming transaction, craft a new transaction and then pay to place it before that of the victim.

[1] Competition through higher transaction fees occurs via "gas replacement" transactions, whereby a pending transaction is resubmitted with a higher fee. The resulting game is akin to an auction (see [3]). The most popular way to make side payments to validators is to use flashbots (see `https://github.com/flashbots/pm`).

[2] Whether it is the most efficient to achieve this goal is a different issue we do not address here.

In this paper, we propose a game-theoretic model of front-running. We use it to study a simple commit-reveal protocol that can be implemented at the smart contract level without modifying the underlying Ethereum infrastructure or introducing third parties (or layer-2 networks). We derive conditions under which an honest player is better off using the protocol than Ethereum's standard procedure. On the cost side, the protocol requires sending two messages instead of one and imposes a delay. Hence, if the cost of sending messages or waiting is high, the protocol is worse than the standard way to send transactions; if they are low, the protocol is preferred. On the benefit side, the protocol can eliminate front-running attacks, especially when it is difficult for an attacker to guess, that is, when the expected payoff of an attacker who commits without knowing whether the victim committed and what message was committed is low. We also argue that our protocol does not impede legitimate competition between honest users, that is, competition to have a transaction included earlier in a block between users who do not rely on observing each other's message.

Prior work

Our commit-reveal protocol is novel but similar to existing proposals. Our main contribution is the type of analysis. In particular, we show that our protocol can reduce front-running attacks while maintaining legitimate users' competition. Existing solutions instead are either primarily concerned with eliminating attacks (at the cost of also eliminating legitimate competition, see, for example, Aequitas protocol and the hedera-hashgraph project) or better organizing competition (at the cost of exacerbating attacks, see, for example, Flashbots). Furthermore, most of the literature has proposed solutions to reduce or eliminate front-running in Ethereum by changing its infrastructure or introducing third parties (See [5] for a review of the literature). Instead, our solution does not require third parties and can be implemented at the smart contract level, allowing for flexibility in its implementation.

With respect to existing solutions, our protocol can be seen as a simplified version of the submarine commitments protocol in [1]: in both cases, a message is first committed and then revealed, and the commitment can be hidden in the sense that the identity of the sender and receiver of the commit message cannot be observed. The main difference is that we adopt a weaker notion of "commitment" because we allow users not to send a transaction after committing it. The notion of "commitment" in [1] is instead stronger because users are penalized for not following through with their commitment.

As already mentioned, we provide a game-theoretic analysis of the properties of this protocol, applicable to any smart contract.[3] With this respect, our work is inspired by [4], who develop a game-theoretic analysis of the problem of front-running arising when an honest user and an attacker claim the same reward. They also propose a protocol that eliminates these types of attacks. Their key assumption is that the legitimate claimant strictly prefers the reward to be burned rather than paid to the attacker. Therefore, these results are useful in some environments where front-running may emerge, but not all. For example, front-running attacks are a serious concern in the AMMs, but in this context, it may not be possible to "burn the reward".

[3] [1] analyze the properties of the submarine commitment scheme in the context of a bug-bounty scheme they propose.

2 The problem: front-running attacks

We start by developing a simple model of front-running attacks and later introduce the commit-reveal protocol.[4] There is a smart contract SC and two players: $Alice$ and Bob. Absent front-running attacks, player A sends a message $\sigma_A \in \Sigma$ to the mempool (i.e., the set of pending transactions), where Σ is the set of possible messages that A may send. When the message σ_A is included in a block, the smart contract SC performs an action that generates a benefit P_A to player A.[5] Front-running attacks arise because messages in the mempool are public. Hence, after A sends a message to the mempool, this message is observed by B, who can send a counter-message $\sigma_B \in \Sigma$. If σ_B is included in the blockchain before A's message, then B earns $P_B(\sigma_A)$ while A earns nothing. Else, B earns nothing and A earns P_A.

Sending messages is costly. Each player can send a regular message by paying $c > 0$. If multiple regular messages are sent, they are included in the block in the order they are sent. Player B, however, can also pay $f > c$ to send a "fast" message that, with probability q, is included in the block before A's regular message, despite A's message being sent first. For example, f could be the cost of sending a transaction via a service such as flashbots, or could be a regular mempool transaction with a transaction fee significantly above the base fee. We consider the parameters q, c, and f as exogenous and determined by the technology available to A and B.

Equilibrium

We can easily solve the game by backward induction and assuming that each player maximizes his/her expected payoff. If A sends a message, then B attempts to front-run if and only if:

$$qP_B(\sigma_A) > f$$

Given this, we can derive A's optimal strategy. Suppose that $qP_B(\sigma_A) < f$, so that A expects no front running. In this case, she sends a message if and only if

$$P_A > c$$

If, instead, $qP_B(\sigma_A) > f$, then A anticipates that B will try to front-run. In this case, A sends a message if and only if

$$(1 - q)P_A > c$$

Hence, front running does not happen when its benefit is low (i.e., $P_B(\sigma_A) \leq f/q$). If, instead, its benefit is large (i.e., $P_B(\sigma_A) > f/q$), B will attempt to front run A whenever A sends a message. In particular, when $P_A > c$ but $(1 - q)P_A < c$ the threat of front running prevents A from sending the message in the first place, therefore destroying the value of the exchange between A and SC.

3 Preventing front-running via commitment

We now use the model developed in the previous section to study how a commit-reveal protocol can mitigate front-running attacks. In terms of notation, we call player A's commit message $\sigma_{A,1}$ and reveal message $\sigma_{A,2}$. Similarly, player B's counter-messages are $\sigma_{B,1}$ and $\sigma_{B,2}$.

[4] For a more detailed analysis, see [2].
[5] For simplicity, here we assume that P_A is independent on the message σ_A. See [2] for the case in which A's payoff depends on her message.

Formally, the protocol has a commitment period and a reveal period, which here are two subsequent blocks. If player A wants to send message $\sigma_A \in \Sigma$ to SC, in the commit period A sends the commit message

$$\sigma_{A,1} = S(addr, \sigma_A)$$

to SC where $addr$ is an address that A controls and $S()$ is a function with an intractable pre-image problem (for example, $Hash\,(addr|\sigma_A)$ where $Hash()$ is the SHA-256 hash function). Once the commit message is included in a block, A sends the reveal message $\sigma_{A,2} = \sigma_A$ to SC from the address $addr$, which is then included in the next block. Upon receiving the message, SC computes $S(addr, \sigma_A)$ and checks whether it received message $S(addr, \sigma_A)$ in the previous block.

It follows that if B wants to front run A he will need to commit a message at the commit stage and then reveal it at the reveal stage. There is a common discount factor $\beta \in [0, 1]$, so when a given payoff is earned with a block delay, this payoff is discounted by β. Finally, A does not observe B's commit message and hence cannot detect B's attempt to front running. At the same time, we assume B observes A's commit message.

3.1 Equilibrium

The first, rather immediate, result is that there is no equilibrium in which B sends the same commit message as A. To see this, suppose that player A sends the commit message $S(addr, \sigma_A)$ and player B sends the same commit message. If in the next period B sends the message $reveal_B = \sigma_A$, then the SC will consider B's reveal message as invalid because sent from an address different from $addr$. It is also easy to see that there is no equilibrium in which A commits but then does not reveal because A can do better by not committing at all. The next lemma summarizes these observations.

▶ **Lemma 1** (No cloning in equilibrium). *There is no equilibrium in which $\sigma_{B,1} = \sigma_{A,1}$. There is also no equilibrium in which A sends the commit message but not the reveal message.*

In equilibrium, therefore, if B wants to attack, he needs to craft a commit message while being completely uninformed about the contents of A's message. However, B anticipates that he will observe A's message and, at that point, will decide whether or not to send the message he initially committed. Therefore, the protocol severely limits but does not fully eliminate B's ability to act upon his observation of A's message.

Formally, suppose $\sigma_{A,1} \neq \emptyset$ (so that A sent the commit message), B committed a message with content σ_B and then observed A's reveal message. In this case, B's expected payoff from front-running is

$$q \cdot P_B(\sigma_B, \sigma_A) - f.$$

Hence, B will try to front run if and only if $q \cdot P_B(\sigma_B, \sigma_A) > f$.

In the commitment phase, B's choice of what message to commit is made in anticipation that he will decide to front run after observing A's reveal message. We assume that B has a prior belief over what message A may send. His expected future payoff is, therefore:

$$\pi \equiv \max_{\sigma_B \in \Sigma} E_{\sigma_A} \left[\max\{q \cdot P_B(\sigma_B, \sigma_A) - f, 0\} | \sigma_{A,1} \neq \emptyset \right],$$

where the expectation is with respect to σ_A. Hence, if A sends a commit message and B tries to front run, B's expected payoff is $\beta\pi - c$. We, therefore, have the following proposition:[6]

[6] The existence of the equilibrium follows from the fact that the players' strategy space is finite, as noted already in [6].

▶ **Proposition 2.** *If $\pi \leq \frac{c}{\beta}$ (i.e., "guessing is hard for B"), then there is no front-running in equilibrium. If instead $\pi > \frac{c}{\beta}$ (i.e., "guessing is easy for B"), front running occurs with strictly positive probability in equilibrium.*

Note that in case "guessing is easy for B", there could be a pure strategy equilibrium in which B commits with probability 1 whenever A commits, or a mixed strategy equilibrium in which B commits with some probability. In either case, after committing, B attempts to front-run A or not depending on A's reveal message.

It is easy to check that in the "guessing is hard for B" case, A's equilibrium payoff is

$$\max\{-c + \beta(P_A - c), 0\}$$

Therefore, the protocol generates both costs and benefits for player A. The benefit is that the simple commit-reveal protocol effectively dissuades B from attacking, yet this comes at a cost: one additional message is required, and the payoff is earned with a one-block delay (and hence is discounted by the parameter β).

4 Conclusion

We conclude by informally discussing two properties of the commit-reveal protocol. As already mentioned, π measures "how easy" it is for B to guess what message he should commit. Therefore, it measures how much, absent the commit-reveal protocol, B relies on observing A's message to attack. At an intuitive level, it can be interpreted as a proxy for the severity of a front-running attack: high values of π imply less severe attacks because B is already informed and relies less on observing σ_A; low values of π imply less severe attacks because B is uninformed and relies heavily on observing σ_A. Therefore, our protocol is most effective at preventing the most severe front-running attacks.

Finally, it is also possible that π is so large that B always wants to commit and reveal a message, whether he observes A's commit message or not. In this case, B acts more like a legitimate competitor because he would commit even if he were to move first. In this case, our commit-reveal protocol preserves competition because both A and B commit their messages and then compete in the reveal stage to have their message included earlier in the block.

References

1 Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

2 Andrea Canidio and Vincent Danos. Commitment against front running attacks, 2023. `arXiv:2301.13785`.

3 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint*, 2019. `arXiv:1904.05234`.

4 Joshua S Gans and Richard T Holden. A solomonic solution to ownership disputes: An application to blockchain front-running. Technical report, National Bureau of Economic Research, 2022.

5 Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint*, 2022. `arXiv:2203.11520`.

6 John Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

The Demand for Programmable Payments: Extended Abstract

Charles M. Kahn
Department of Finance, University of Illinois, Urbana-Champaign, IL, USA

Maarten R.C. van Oordt
Tinbergen Institute, Amsterdam, The Netherlands
Vrije Universiteit Amsterdam, The Netherlands

Abstract

In [1], we examine the desirability of programmable payments, arrangements in which transfers are automatically executed conditional upon preset objective criteria. We study optimal payment arrangements in a continuous-time framework where a buyer and a seller of a service interact. We stack the cards in favor of programmable payments by considering an environment where neither agent has any legal recourse if the other fails to deliver upon their promises. We identify scenarios where programmable payments could improve economic outcomes and scenarios where they cannot. Direct payments increase the surplus by avoiding the liquidity cost of locking-up funds in a programmable payment arrangement until the moment where the conditions are satisfied to release those funds to the payee.

Programmable payments will be desirable, and may in fact be the only viable payment arrangement, in situations where economic relationships are of a short duration. Nonetheless, there is a limit to the length of the arrangement a single programmable payment can support, because eventually the additional liquidity cost of locking up *more* funds for a *longer* period starts to exceed the additional surplus generated from extending the length of the arrangement. For longer periods multiple payments are necessary.

Sufficiently long optimal chain-of-payments arrangements always start with direct payments because of the lower liquidity costs. Only towards the end of a relationship do the parties switch to the use of programmable payments. Moreover, the optimum for infinitely long payment arrangements consists of direct payments only. These results suggest that programmable payments are unlikely to become the new "standard" for all payment arrangements.

Many have argued that technological developments in the payments space will lead to an explosion of so-called micro-payments. Our results suggest a more complex relationship between transactions cost and the number of payments. Lower transaction costs increase the number of payments for the *extensive* margin in the sense of increasing the set of potential buyer-seller pairs where transaction costs are no longer prohibitively expensive. For the *intensive* margin, that is, within buyer-seller pairs, we find the opposite effect: lower transaction costs are associated with fewer payments, as trust becomes easier to achieve.

2012 ACM Subject Classification Applied computing → Digital cash; Applied computing → E-commerce infrastructure; Applied computing → Electronic funds transfer; Applied computing → Online banking; Social and professional topics → Consumer products policy

Keywords and phrases Electronic payment, smart contracts, programmable payment

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2022.8

Category Extended Abstract

Funding *Maarten R.C. van Oordt*: gratefully acknowledges financial support from the Fintech Chair of Université Paris Dauphine – PSL.

References

1 C.M. Kahn and M.R.C. Van Oordt. The Demand for Programmable Payments. *Tinbergen Institute Discussion Paper*, 2022-076, 2022.