Brief Announcement: Highly Dynamic and Fully Distributed Data Structures

John Augustine
Indian Institute of Technology Madras, Chennai, India

Antonio Cruciani
Aalto University, Espoo, Finland

Iqra Altaf Gillani
National Institute of Technology Srinagar, India

Abstract

We study robust and efficient distributed algorithms for building and maintaining distributed data structures in dynamic Peer-to-Peer (P2P) networks. P2P networks are characterized by a high level of dynamicity with abrupt heavy node *churn* (nodes that join and leave the network continuously over time). We present a novel algorithmic framework to build and maintain, with high probability, a skip list for poly(n) rounds despite a *churn rate* of $\mathcal{O}(n/\log n)$, which is the number of nodes joining and/or leaving *per round*; n is the stable network size. We assume that the churn is controlled by an oblivious adversary that has complete knowledge and control of what nodes join and leave and at what time and has unlimited computational power, but is oblivious to the random choices made by the algorithm. Importantly, the maintenance overhead in any interval of time (measured in terms of the total number of messages exchanged and the number of edges formed/deleted) is (up to log factors) proportional to the churn rate. Furthermore, the algorithm is scalable in that the messages are small (i.e., at most polylog(n) bits) and every node sends and receives at most polylog(n) messages per round.

To the best of our knowledge, our work provides the first-known fully-distributed data structure and associated algorithms that provably work under highly dynamic settings (i.e., high churn rate that is near-linear in n). Furthermore, the nodes operate in a localized manner.

Our framework crucially relies on new distributed and parallel algorithms to merge two n-element skip lists and delete a large subset of items, both in $\mathcal{O}(\log n)$ rounds with high probability. These procedures may be of independent interest due to their elegance and potential applicability in other contexts in distributed data structures.

Finally, we believe that our framework can be generalized to other distributed and dynamic data structures including graphs, potentially leading to stable distributed computation despite heavy churn.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Networks → Peer-to-peer networks; Theory of computation → Data structures design and analysis

Keywords and phrases Peer-to-peer network, dynamic network, data structure, churn, distributed algorithm, randomized algorithm

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.47

Related Version *Full Version*: `https://arxiv.org/abs/2409.10235` [2]

Funding *Antonio Cruciani*: This work was supported in part by the Research Council of Finland, Grant 363558 and Partially supported by the Cryptography, Cybersecutiry, and Distributed Trust laboratory at IIT Madras (Indian Institute of Technology Madras) while visiting the institute.

1 Introduction

Peer-to-peer (P2P) networks underpin many modern distributed systems for resource sharing, storage, messaging, and content distribution (e.g., Gnutella, Skype, BitTorrent, Signal). They are highly dynamic: nodes join and leave continuously (*churn*), and links change arbitrarily,

leading to frequent topological disruptions. Measurements of real-world P2P networks (see for example [10]) show that up to 50% of peers may be replaced within an hour, yet network size often remains relatively stable.

P2P systems support diverse tasks such as data storage and retrieval [16, 9], collaborative filtering [7], spam and malware detection [8, 18], and privacy-preserving data management [11]. Efficient distributed data structures such as distributed hash tables (e.g., CAN, Chord, Pastry, Tapestry) achieve good load balancing but offer little control over data placement and limited resilience to failures. Structured overlays like Skip Graphs [1], SkipNets [14], Rainbow Skip Graphs [12], and Skip$^+$ [15] allow ordered search and have been formally shown to be resilient to a limited number of faults (or equivalently small amounts of churn).

However, none of these data structures have theoretical guarantees of being able to work in a dynamic network with a very high adversarial churn rate, which can be as much as near-linear (in the network size) per round. This is a fundamental limitation for dynamic large-scale P2P systems, where it is essential to (i) preserve structural integrity, (ii) update quickly after joins and deletions, and (iii) answer queries reliably, all while keeping maintenance work proportional to the churn. Rebuilding from scratch is too costly; maintaining robustness under heavy churn without sacrificing simplicity or scalability remains an open challenge.

In this work, we address this gap by designing the first distributed skip list with rigorous guarantees under near-linear churn, ensuring fast updates, query correctness, and resource-competitive maintenance.

2 Preliminaries

Model: Dynamic Networks with Churn. We adopt the *dynamic network with churn* (DNC) model [4, 3]: a synchronous message-passing network of fixed size n, controlled by an *oblivious* adversary that may replace up to $\mathcal{O}(n/\log n)$ nodes per round after an initial $B = \beta \log n$-round *bootstrap phase* with no churn. Each node has a unique ID in $[0, poly(n)]$, stores one item, and can send/receive at most polylog(n)-bit messages over at most polylog(n) links per round. New nodes are connected to distinct existing nodes to avoid congestion. The adversary specifies a sequence $(V_t)_{t\geq 0}$ with $|V_t| = n$ for all t; joins and leaves are arbitrary subject to the churn bound. Nodes may create/delete edges; a bidirectional link (u, v) exists if both endpoints exchange invitation/acceptance messages, and is removed when either endpoint leaves or sends a delete message. For clarity, we focus here on the fixed-n, single-item case; in the full version, we generalize our algorithms to handle multiple items per node and a network size that may vary over time [2].

Problem Statement. We aim to maintain an *approximate distributed data structure* storing all items in the network, resilient to high adversarial churn. Insertions and deletions complete in $O(\log n)$ rounds, and membership queries $q(x, r, s)$ (initiated at source s in round r) must be answered within $Q = \mathcal{O}(\log n)$ rounds. A query returns "Yes" if x is present for the entire interval $[r, r + Q]$, "No" if absent for the entire interval, and may return either answer otherwise. This relaxed consistency, or *eventual correctness*, allows scalability and liveness under churn.

We require a *dynamic* formulation of *resource-competitiveness* [6]: the total work (number of messages and edge addition/deletions) over any interval is within a polylog(n) factor of the churn in that interval (allowing an $\mathcal{O}(\log n)$ lookback to capture delayed effects).

We fist show how to maintain an approximate distributed skip list, and then we extend our techniques to skip graphs and, in the full version, to general pointer-based distributed data structures.

3 Our Contribution

We first present our main result: the design and analysis of a churn-resilient distributed skip list with provable guarantees under near-linear adversarial churn. We then show how our techniques extend to more complex structures, including skip graphs, and, more generally, any pointer-based distributed data structure; we refer to the extended version for full details [2].

Main Result. Our dynamic distributed skip list data structure is architected using multiple "virtual networks" (see Figure 1). Each peer node can participate in more than one network and in some cases more than one location within the same network. We use the following network structures.

The Spartan Network $\mathcal{S}$ is a wrapped butterfly network that contains all the current nodes. This network can handle heavy churn of up to $\mathcal{O}(n/\log n)$ nodes joining and leaving in every round [5]. However, this network is not capable of handling search queries.

Live Network $\mathcal{L}$ is the skip list network on which all queries are executed. Some of the nodes in this network may have left. We require such nodes to be temporarily represented by their replacement nodes (from their respective neighbors in $\mathcal{S}$).

Buffer Network $\mathcal{B}$ is a skip list network on which we maintain all new nodes that joined recently.

Clean Network $\mathcal{C}$ is a skip list network that seeks to maintain an updated version of the data structure that includes the nodes in the system.

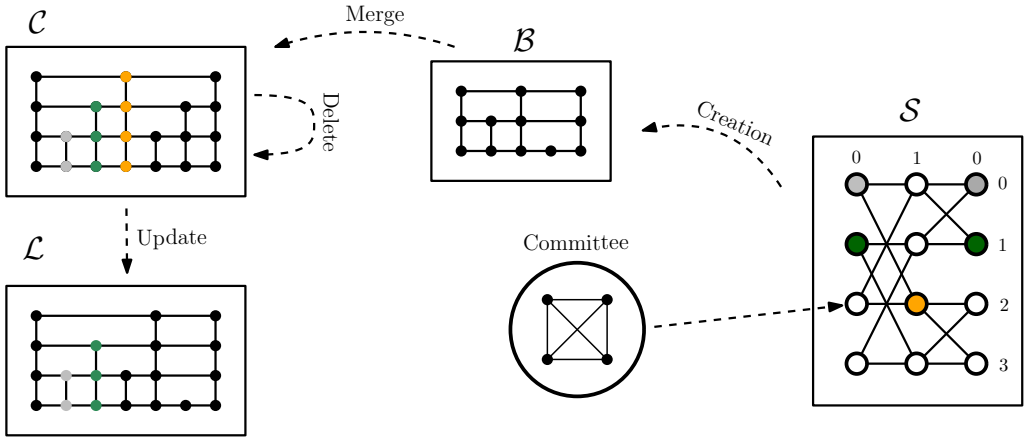

Figure 1 Schematic representation of the architecture and the Maintenance cycle described in Algorithm 1. Colored nodes in $\mathcal{L}$ and $\mathcal{C}$ are nodes that have been removed by the adversary and that are being covered by some committee of nodes (of the same color) in $\mathcal{S}$.

Moreover, in all skip lists, when a node exits the system, it is operated by a selected group of nodes. In the course of our algorithm description, if a node u is required to perform some operation, but is no longer in the system, then its replacement node(s) will perform that operation on its behalf. Note further that some of the replacement nodes themselves may need to be replaced. Such replacement nodes will continue to represent u. The protocol assumes a short ($\Theta(\log n)$ round) initial "bootstrap" phase, where there is no churn[1] and it

[1] Without a bootstrap phase, it is easy to show that the adversary can partition the network into large pieces, with no chance of forming even a connected graph.

initializes the underlying network. More precisely, the bootstrap is divided in two sub-phases in which we (i) build the underlying churn resilient network in $\mathcal{O}(\log n)$ rounds and, (ii) we build the skip lists data structures $\mathcal{L}$ and $\mathcal{C}$ (initially $\mathcal{L} = \mathcal{C}$) in $\mathcal{O}(\log n)$ rounds; after this, the adversary is free to exercise its power to add or delete nodes up to the churn limit and the network will undergo a continuous maintenance process. The overall maintenance of the dynamic distributed data structure goes through cycles. Each cycle $c \geq 1$ in Algorithm 1 consists of four sequential phases. Without loss of generality, assume that initially $\mathcal{L}$ and $\mathcal{C}$ are the same. We use the notation $\mathcal{L}^{(c)}$ (resp. $\mathcal{S}^{(c)}, \mathcal{C}^{(c)}, \mathcal{B}^{(c)}$) to indicate the network $\mathcal{L}$ (resp. $\mathcal{S}, \mathcal{C}, \mathcal{B}$) during the cycle $c \in \mathbb{N}$, when it is clear from the context we omit the superscript to maintain a cleaner exposition.

Algorithm 1 Overview of the distributed skip list maintenance process.

1 Starting from a skip list $\mathcal{L}^{(1)} = \mathcal{C}^{(1)}$
2 **foreach *Cycle*** $c \geq 1$ **do**
3 **Phase 1 (Deletion):** All the replacement nodes in $\mathcal{C}^{(c)}$ are removed. Note that nodes that leave the system during the replacement process may remain in $\mathcal{C}^{(c)}$.
4 **Phase 2 (Buffer Creation):** All the new nodes that were churned into the system since Phase 2 of the previous cycle join together to form $\mathcal{B}^{(c)}$.
5 **Phase 3 (Merge):** The buffer $\mathcal{B}^{(c)}$ created in Phase 2 is merged into $\mathcal{C}^{(c)}$, i.e., $\mathcal{C}^{(c+1)} \leftarrow \mathcal{C}^{(c)} \cup \mathcal{B}^{(c)}$.
6 **Phase 4 (Update):** Update the live network $\mathcal{L}^{(c)}$ with the clean network $\mathcal{C}^{(c)}$, i.e., $\mathcal{L}^{(c+1)} \leftarrow \mathcal{C}^{(c+1)}$.
7 **end**

Each phase of Algorithm 1 requires $\mathcal{O}(\log n)$ rounds w.h.p. In particular, the deletion and merge phases implement two novel parallel and fully distributed protocols to delete a batch of $\Theta(n)$ and merge two skip lists of $\Theta(n)$ elements in $\mathcal{O}(\log n)$ rounds w.h.p, respectively. This improves over prior $\mathcal{O}(k \log n)$-round bounds to perform these operation on a batch of k elements in skip-lists. We believe that some ideas from our results could be used in the centralized batch parallel setting to quickly insert batches of new elements in skip lists-like data structures (see e.g.,[17]) and in the fully dynamic graph algorithms settings (see for example the survey [13]) to perform fast updates of fully dynamic data structures. Indeed, in principle (provided the right amount of parallelism), our deletion and merge algorithms could be implemented in a parallel (centralized) setting and used to speed-up all kinds of computations involving these specific data structures. Our major contribution can be summarized in the following theorem.

▶ **Theorem 1** (Main Theorem). *Given a dynamic set of peers initially connected in some suitable manner (e.g., as a single path) that is stable for an initial period of $\mathcal{O}(\log n)$ rounds (i.e., the so-called* bootstrap phase*) and subsequently experiencing heavy adversarial churn at a churn rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round, there exists a resilient skip list, a distributed data structure that can withstand heavy adversarial churn at a rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round. We provide the following algorithms to support this data structure.*

- *An $\mathcal{O}(\log n)$ round algorithm to construct the resilient skip list during the bootstrap phase,*
- *a fully distributed algorithm that maintains the resilient skip list as nodes are inserted into or deleted from the data structure with the guarantee that the data structure reflects the updates within $\mathcal{O}(\log n)$ rounds,*

- *and a membership query algorithm that any peer can invoke and can be answered with efficiency parameter $Q \in \mathcal{O}(\log n)$.*

All nodes send and receive at most $\mathcal{O}(polylog(n))$ messages per round, each comprising at most $\mathcal{O}(polylog(n))$ bits. Moreover, such a maintenance algorithm is dynamically resource competitive. The maintenance protocol ensures that the resilient skip list is maintained effectively for at least n^d rounds with high probability, where $d \geq 1$ can be an arbitrarily chosen constant.

The above architecture and results extend to skip graphs and, to settings with multiple items per node and variable network size.

▶ **Corollary 2.** *Given a network with n nodes in which each vertex v possesses $t = polylog(n)$ elements in the skip list. Then maintenance protocol requires $\mathcal{O}(\log n)$ rounds to build and maintain a resilient skip list that can withstand heavy adversarial churn at a churn rate of up to $\mathcal{O}(n/\log n)$ nodes joining/leaving per round.*

▶ **Corollary 3.** *The maintenance cycle described in Algorithm 1 can be adapted to support skip graphs. In particular, the resulting data structure remains resilient to churn at a rate of up to $\mathcal{O}(n/\log n)$ per rounds, and all operations complete within $\mathcal{O}(\log n)$ rounds with high probability, using at most $\mathcal{O}(polylog(n))$ messages per node per round.*

A General Framework for Churn-Resilient Structures. On a broader picture, the above results can be unified under a single general framework for maintaining pointer-based data structures under adversarial churn. Given a data structure $\mathcal{D}$, we can "abstract away" the ideas in Algorithm 1 and obtain the following *abstract maintenance cycle* for a generic data structure $\mathcal{D}$:

1. **Delete Phase:** Identify and remove corrupted, outdated, or disconnected regions of $\mathcal{D}$. Deletion ensures that inconsistencies caused by churn do not propagate through the structure.
2. **Creation Phase:** Organize the set of newly arrived nodes into a provisional structure $\mathcal{B}$ (the buffer data structure). The goal is to prepare these nodes for integration, typically by arranging them into a sorted or partially structured form.
3. **Merge Phase:** Integrate the buffer network $\mathcal{B}$ into the main structure $\mathcal{D}$ using a distributed merge protocol. This phase reconstructs the structure while respecting existing invariants.
4. **Update Phase:** Perform any necessary local corrections, including pointer rebalancing, level adjustments, or redundancy restoration, to finalize the integration.

Provided that we have $\mathcal{O}(T)$-round distributed algorithms for each phase of the above abstract maintenance cycle, we can maintain $\mathcal{D}$ against an adversarial churn rate of $\mathcal{O}(n/T)$ per round.

▶ **Theorem 4.** *Let $\mathcal{D}$ be a distributed pointer-based data structure maintained using our four-phase cycle, and let T be the maximum number of rounds needed for any phase. Then $\mathcal{D}$ can tolerate an adversarial churn rate of up to $\mathcal{O}(n/T)$ nodes per round, while preserving global correctness for at least n^c rounds w.h.p., for any fixed constant $c > 0$.*

Our abstract maintenance cycle allows to define classes of churn-resilient data structures in the DNC model, where each class is characterized by the churn rate per round (thus the maintenance cycle runtime) that the data structure can tolerate while still supporting efficient update and query operations. Formally, let t be the maintenance cycle run time function. We say that a distributed data structure $\mathcal{D}$ is **t-maintainable** if it can be successfully maintained by a t-round maintenance cycle.

▶ **Observation 5.** *Under the above general framework, every t-maintainable distributed data structure tolerates an adversarial churn rate of $\mathcal{O}(n/t)$ per round.*

In this brief announcment, we showed that Skip-Lists and Skip-Graphs are both $\mathcal{O}(\log n)$-maintainable and that are robust against and oblivious adversarial churn rate of $\mathcal{O}(n/\log n)$ per round. We conjecture that our bounds are tight, in the sense that $\log n$ is also a lower bound for the maintenance cycles for these data structures. That is because, in order to beat the $\Omega(\log n)$ barrier we would need to solve distributed sorting faster than in $\log n$ rounds while maintaining dynamic resource competitiveness.

This classification also raises a number of intriguing open problems. For example, it remains unclear whether there exist distributed data structures in the DNC model that are $\log\log n$-maintainable, and more generally, how to characterize the precise boundaries between maintainability classes. A key challenge is to establish lower bounds on the maintenance cycle time for fundamental distributed data structures. Do entirely new data structures need to be designed to exploit faster maintenance cycles? We believe that our formulation of maintainability classes in the DNC model opens up a rich landscape for further exploration.

4 Concluding remarks and discussion

In this work we proposed the first churn resilient skip list that can tolerate a heavy adversarial churn rate of $\mathcal{O}(n/\log n)$ nodes per round. The data structure can be seen as a four networks architecture in which each network plays a specific role in making the skip list resilient to churns and keeping it continuously updated. Moreover, we provided efficient $\mathcal{O}(\log n)$ rounds resource competitive algorithms to (i) delete a batch of elements from a skip list (ii) create a new skip list and, (iii) merge together two skip lists. This last result is the first algorithm that can merge two skip lists (as well as a skip list and a batch of new nodes) in $\mathcal{O}(\log n)$ rounds w.h.p.. We point out that these algorithms can be easily adapted to work on skip graphs [1, 12, 15].

In a broader sense, our technique is general and can be seen as a framework to maintain any kind of distributed data structure despite heavy churn rate. The only requirement is to devise efficient *delete, buffer creation, merge*, and *update* algorithms for the designated data structure. Furthermore, this allows us to define complexity classes for the maintenance of distributed data structure in the Dynamic Networks with Churn Model. Indeed, in this work we showed that skip list and skip graphs belong to the class of data structure that can tolerate a churn rate of $\mathcal{O}(n/\log n)$ per round.

Finally, given the simplicity of our approach, we believe that our algorithms could be used as building blocks for other non-trivial distributed computations in dynamic networks.

References

1 James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, 2007. doi:10.1145/1290672.1290674.

2 John Augustine, Antonio Cruciani, and Iqra Altaf Gillani. Maintaining distributed data structures in dynamic peer-to-peer networks, 2024. doi:10.48550/arXiv.2409.10235.

3 John Augustine, Gopal Pandurangan, Peter Robinson, Scott T. Roche, and Eli Upfal. Enabling robust and efficient distributed computation in dynamic peer-to-peer networks. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.29.

4 John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the Twenty-Third*

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012. SIAM, 2012. doi:10.1137/1.9781611973099.47.

5 John Augustine and Sumathi Sivasubramaniam. Spartan: Sparse robust addressable networks. *J. Parallel Distributed Comput.*, 2021. doi:10.1016/J.JPDC.2020.12.013.

6 Michael A. Bender, Jeremy T. Fineman, Mahnush Movahedi, Jared Saia, Varsha Dani, Seth Gilbert, Seth Pettie, and Maxwell Young. Resource-competitive algorithms. *SIGACT News*, 2015. doi:10.1145/2818936.2818949.

7 John F. Canny. Collaborative filtering with privacy. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*. IEEE Computer Society, 2002. doi:10.1109/SECPRI.2002.1004361.

8 Inc Cloudmark. Website of cloudmark. https://cloudmark.com/, last checked on 14-February-2024.

9 Peter Druschel and Antony I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20-23, 2001, Elmau/Oberbayern, Germany*. IEEE Computer Society, 2001. doi:10.1109/HOTOS.2001.990064.

10 Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas E. Anderson. Profiling a million user dht. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007*. ACM, 2007. doi:10.1145/1298306.1298325.

11 Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings*. USENIX Association, 2009. URL: http://www.usenix.org/events/sec09/tech/full_papers/geambasu.pdf.

12 Michael T. Goodrich, Michael J. Nelson, and Jonathan Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109601.

13 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 2022. doi:10.1145/3555806.

14 Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *4th USENIX Symposium on Internet Technologies and Systems, USITS'03, Seattle, Washington, USA, March 26-28, 2003*. USENIX, 2003. URL: http://www.usenix.org/events/usits03/tech/harvey.html.

15 Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip$^+$: A self-stabilizing skip graph. *J. ACM*, 2014. doi:10.1145/2629695.

16 Antony I. T. Rowstron and Peter Druschel. Storage management and caching in past, A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating System Principles, SOSP 2001, Chateau Lake Louise, Banff, Alberta, Canada, October 21-24, 2001*. ACM, 2001. doi:10.1145/502034.502053.

17 Thomas Tseng, Laxman Dhulipala, and Guy E. Blelloch. Batch-parallel euler tour trees. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*. SIAM, 2019. doi:10.1137/1.9781611975499.8.

18 Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. Security applications of peer-to-peer networks. *Comput. Networks*, 2004. doi:10.1016/J.COMNET.2004.01.002.