GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

Doctoral Thesis

Models and Algorithms for Temporal Betweenness Centrality and Dynamic Distributed Data Structures

PhD Program in Computer Science: XXXVI cycle

Supervisor:

Prof. Francesco Pasquale
pasquale@mat.uniroma2.it

Author:

Antonio Cruciani
antonio.cruciani@gssi.it

co-Supervisor:

Prof. Pierluigi Crescenzi
pierluigi.crescenzi@gssi.it

February 2025

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

To Adelya

Abstract

Networks are ubiquitous mathematical objects for modeling a wide range of real-world systems, from social networks and communication infrastructures to biological processes such as protein interactions. A distinctive feature of these real-world networks is that they evolve over time, with nodes and edges dynamically changing at any time. This temporal evolution introduces several new challenging problems, ranging from the choice of the "right" evolving network model that must be considered to the design and analysis of both centralized and distributed algorithms in many scenarios. In this thesis, we start by considering *temporal graphs*, where interactions are timestamped. They offer a rich model for capturing the dynamics of evolving networks from a centralized perspective, but they also introduce new computational challenges, since traditional static graph algorithms often cannot be directly applied to temporal data. Designing efficient centralized algorithms for temporal graphs requires dealing with both the structural and temporal dimensions of the network, while ensuring that solutions are not only accurate but also scalable. For instance, temporal centrality metrics, such as *temporal betweenness centrality*, must account for the ordering and timing of interactions. With respect to the static case, this complicates their computation but it provides more insightful results about the importance of the nodes in the network during time. In particular, temporal betweenness assigns a value to each node that is based on the fraction of optimal temporal paths passing through it. Buß et al. (KDD, 2020) gave algorithms to compute various notions of temporal betweenness centrality, including perhaps the most natural one –*shortest temporal betweenness*. Their algorithm computes the centrality values of all nodes in time $\mathcal{O}(n^3T^2)$, where n is the size of the network and T is the total number of time steps. For real-world networks, which easily contain tens of thousands of nodes, this complexity becomes prohibitive. Thus, it is reasonable to consider fast approximation algorithms that allow for measuring the relative importance of nodes in very large temporal graphs. In this thesis, we consider the problem of efficiently computing the temporal betweenness rankings and scores. We start by considering proxies (i.e., fast heuristics) to approximate the temporal betweenness rankings that take into account *global* and *local* properties of the network. We compare several such proxies on a diverse set of real-world networks. To this end we define a novel temporal degree notion called the *pass-through-degree*, which measures the number of pairs of neighbors of a node that are temporally connected through it, and we show that such a temporal degree notion can be computed in nearly linear time for all nodes. Moreover, we observe that it is surprisingly competitive as a proxy for the temporal betweenness centrality. Next, we consider the problem of approximating the temporal betweenness scores of all the nodes in the network. To this end, we develop a novel sampling-based approximation

algorithm that computes probabilistically guaranteed high-quality temporal betweenness estimates (of nodes and temporal edges). Such an algorithm uses advanced tools from statistical learning theory and combinatorial optimization to estimate the temporal betweenness of all the nodes up to a small absolute error ε with probability of at least $1 - \delta$, where $\varepsilon, \delta \in (0, 1)$. We empirically show how the proposed algorithm achieves tight theoretical guarantees and significantly improves the state-of-the-art in terms of running time, approximation quality, sample size, and allocated memory, enabling very precise approximations on very big temporal graphs.

In the second part of the thesis we consider *dynamic network* models from a distributed computing perspective, in which designing algorithms that provably work in highly dynamic environments presents its own set of non-trivial challenges. Networks undergoing frequent changes due to node churn (node leaving and joining the network), rapid topology shifts, or adversarial behaviors demand decentralized solutions that can adapt in real time. These challenges emerge mostly in peer-to-peer systems, where the network structure is constantly evolving. We study the fundamental problem of performing computations in a distributed system subject to a heavy churn rate (i.e., high number of nodes joining and leaving the network in each round). We begin by extending the model by Becchetti et al. (SODA, 2020) to obtain dynamic random graph models that evolve forever: in the first model, edges can be faulty, i.e., each edge at each round disappears with some probability; in the second one, at every round new nodes join the network according to a stochastic process and each node currently in the network disappears with certain probability; in the third one, we consider a combination of the two models above, in which edges can be faulty and nodes can join and leave the network. We run extensive simulations to measure how long it takes a message starting at a random node to reach all, or almost all, the nodes. The simulations show that, for large ranges of the parameters of the models, the information spreads very fast, i.e., at a rate compatible with a logarithmic growth, as a function of the number of nodes in the network. We continue by studying robust and efficient distributed algorithms for building and maintaining distributed data structures in dynamic graphs on which nodes are continuously replaced by an oblivious adversary. We present a novel algorithm that builds and maintains with high probability a skip list for $poly(n)$ rounds despite $\mathcal{O}(n/\log n)$ adversarial churn *per round* (n is the "stable" network size). Our algorithm requires a maintenance overhead proportional to the churn rate. In addition, our algorithm is scalable in the sense that messages are small and every node sends a limited amount of messages in each round. A consequence of our maintenance algorithm is that it opens up opportunities for more general distributed computation in distributed dynamic graph models.

Acknowledgments

I would like to express my gratitude to everyone who has supported and guided me throughout this incredible journey. Without their encouragement, expertise, and support this achievement would not have been possible.

First and foremost, I would like to thank my advisors, Francesco Pasquale and Pierluigi Crescienzi, for their invaluable guidance, encouragement, patience, and belief in my potential.

I am deeply grateful to John Augustine for his invaluable mentorship during my visiting period at IIT Madras. John, you taught me so much, both professionally and personally. From the inspiring projects we worked on together to the precious moments shared over chai while discussing research, your mentorship has profoundly impacted my journey. The vibrant campus environment you introduced me to and the amazing people I met there made my visit unforgettable. Your guidance has been pivotal in shaping the researcher I am today.

I would like to extend my thanks to Shay Kutten and David Peleg for their inspiring collaborations and valuable insights. Working with such accomplished researchers has been a privilege and an immensely rewarding experience.

My sincere gratitude also goes to Leonardo Pellegrina. After meeting at ECML-PKDD and discussing research ideas, our collaboration quickly took shape. Your decision to host me in Padua for further discussions and work was incredibly generous, and our collaboration has been both enjoyable and intellectually stimulating.

I would also like to thank Simran for making my time in India truly memorable. Your guidance and our trips around Chennai exploring the rich culture and amazing food added a unique dimension to my experience.

I am profoundly grateful to my collaborators–Manish, Iqra, Thorsten, Yuval, Gianni, Daniele, Paola, Simone, Ruben and Bojana–for their invaluable insights and contributions, which have greatly enhanced my research.

To all my colleagues at GSSI, IIT-Madras and the University of Rome “Tor Vergata”. My special thanks extend to Girish, Rohin, Mario, Adiel, Riccardo, and Mirko–thank you for the engaging discussions and shared experiences that have made this journey unique.

On a personal note, my work is dedicated to my wife, Adelya. Your unconditional support and encouragement have been my foundation throughout these years. I am eternally grateful that, through the randomness of life, we met and shared every aspect of these incredible years together. I can’t wait to discover what’s ahead of us.

Finally, I would like to thank my family that always believed in me and supported me throughout all my life. “Infine, vorrei ringraziare la mia famiglia per aver sempre creduto in me e spronato a dare il massimo”.

Contents

List of Figures

List of Tables

Preface

This thesis presents two of the four lines of research carried out, together with my collaborators, during my years as a Ph.D. student. This Preface describes all the lines of research covered during my doctoral journey.

Rigorous and Efficient Algorithms for Temporal Networks

This is one of the research directions investigated in this thesis. We design (possibly) simple and efficient approximation algorithms to solve graph mining problems in large temporal networks.

This research produced the following papers:

- R. Becker, P. Crescenzi, A. Cruciani, B. Kodric. "Proxying Betweenness Centrality Rankings in Temporal Networks." *Full paper* published in 21st International Symposium on Experimental Algorithms, (SEA 2023).
- A. Cruciani. "MANTRA: Temporal Betweenness Centrality Approximation Through Sampling." *Full paper* published in European Conference in Machine Learning and Knowledge Discovery in Databases. Research Track (ECML-PKDD 2024).

Efficient Fully Distributed Algorithms for Dynamic Networks with Churns

This is another research direction investigated in this thesis. We design simple distributed algorithms that provably work in dynamic networks in which nodes join and leave continuously.

This research produced the following papers:

- A. Cruciani, F. Pasquale. "Dynamic graph models inspired by the Bitcoin network-formation process" *Full paper* published in the Proceedings of the 24th International Conference on Distributed Computing and Networking (ICDCN 2023). This work was previously presented as a brief announcement in "Stabilization, Safety, and Security of Distributed Systems" conference (SSS 2022).

- J. Augustine, A. Cruciani, I. A. Gillani. "Maintaining Distributed Data Structures in Dynamic Peer-to-Peer Networks". Research paper submitted to a theoretical computer science conference and available as a preprint on ArXiv.

Rigorous and Efficient Graph Mining Algorithms for Static Networks

We propose fast scalable approximation algorithms to compute static graph properties that become prohibitive to compute on very large networks.

This research produced the following papers:

- G. Amati, A. Cruciani, D. Pasquini, P. Vocca, S. Angelini. "PROPAGATE: A Seed Propagation Framework to Compute Distance-Based Metrics on Very Large Graphs". *Full paper* published in the European Conference in Machine Learning and Knowledge Discovery in Databases. Research Track (ECML-PKDD 2023).
- A. Cruciani. "Fast Estimation of Percolation Centrality". Research paper submitted to a Data Mining conference and available as a preprint on ArXiv.

Distributed Algorithms on Anonymous Networks

We propose and study fundamental distributed computing problems in the Anonymous Congested Clique. A new model inspired by privacy-oriented systems that heavily rely on anonymity.

This research produced the following paper:

- J. Augustine, A. Cruciani, M. Kumar, S. Kutten, D. Peleg. "Partial Gossip in an Anonymous Node-Capacitated Clique". Research paper submitted to a theoretical computer science conference.

Chapter 1

Introduction

Networks are fundamental structures used to model relationships and interactions in a wide variety of real-world systems, ranging from social and communication networks to biological and transportation systems. While traditional network models typically assume static relationships, many real-world systems are inherently dynamic: nodes and edges appear, and disappear over time. Motivated by the need to capture such evolving properties, there has been an increasing interest in defining suitable mathematical models that exhibit richer structures than static graph theory. Moreover, there has been an increasing interest in the study of dynamic or temporal networks theory.

Such innovative models, provide an accurate representation of real-world phenomena that are not static but rather change over time. Thus, considering dynamic or temporal networks instead of static ones offers unique perspectives for understanding complex patterns in systems that change over time. For example, in social networks, interactions between individuals are not only *about* who connects with whom but also *when* these connections occur. Similarly, in communication networks, the temporal order of message exchanges may affect information flow, latency, and overall network performance. Other examples of systems that are inherently dynamic and that can not modeled/captured using static graph theory are transportation networks in which an edge between two nodes represents a route between two bus stations that is available at some predetermined schedules dictated by a time table, and biological networks that represent protein-protein interactions or brain connections.

However, the introduction of an additional dimension (e.g. the time dimension) in these models comes with a high price and new unique challenges. For example, traditional graph-theoretic notions, such as centrality, distance-based metrics, and connectivity must be redefined in the context of dynamic/temporal structures that change over time. A typical example is the non-transitivity of the temporal connectivity, due to the

fact that, because of the temporal constraints, a node u can (temporally) reach a node v and v can (temporally) reach another node w, but u cannot (temporally) reach w. Furthermore, using algorithms or techniques for static networks on these richer structures does not lead to meaningful results and, consequently, it does not capture the unique dynamic essence that each of these networks has. In addition, solving problems on such richer structures, usually, turns out to be computationally demanding or even infeasible (i.e., NP-Hard). For example, counting temporal motifs turns out to be more difficult on temporal graphs. In particular, counting simple temporal stars turns out to be NP-Hard [1]. This contrasts with stars in static graphs, which are generally considered trivial to count (the number of non-induced k-edge stars with center node u is $\binom{d_u}{k}$ where d_u is the degree of u). For this reason, additional effort must be put into carefully designing efficient exact or approximate algorithms that provably work under highly dynamic settings.

Temporal Networks have been proposed as one possible generalization of (static) networks. Informally, temporal networks can be seen as edge-labeled graphs in which the labels are timestamps and they indicate the time instants in which the edges are present in the network. Identifying influential (i.e., important under some specific definition of importance) nodes in a network is arguably one of the most important tasks in graph mining and network analysis. A large variety of centrality measures, all aiming at correctly quantifying a node's importance in the network, have been formulated in the literature. One of the most cited ones is the *betweenness centrality*, formally introduced by Freeman [2] in 1997. The betweenness centrality of a node is defined as the sum, over all pairs of nodes, of the fraction of shortest paths between them that pass through the given node. This centrality measure has been widely studied on static graphs. However, the advent of temporal networks introduced the need for a suitable temporal version of such a centrality measure that considers the temporal aspects of paths in these evolving networks. Indeed, running the static algorithms for the betweenness centrality on the underlying graph of the temporal network would completely ignore all the temporal interactions among nodes and lead to misleading results.

A temporal path can capture the information spreading across a sequence of edges, and this can be useful in many scenarios such as defining centrality measures over nodes (and temporal edges) of the network. A temporal centrality measure captures the importance of a node with respect to some topological (and, in our case, temporal) aspect of the network. Intuitively, high centrality nodes (or edges) play a much more important role in the analyzed network than those with a small centrality score. For example, given a temporal graph of contacts among people, finding the most "influential" node during the outburst of an epidemic disease could allow us to quickly isolate and minimize the damage that the disease might cause to the population. An analogous example can

be formulated for the centrality of the edges of the static *fingerprint* of the temporal graph (i.e., the temporal graph without the time instants on the edges). Indeed, finding the edges with high centrality scores can help (for example) the Ministry of Transport to identify roads that tend to be highly jammed during the day. Thus to wisely and efficiently allocate public resources, and build alternative roads to ease the lives of the citizens that have to commute on these every day. *Temporal Betweenness Centrality* considers the number of optimal temporal paths flowing through a specific node or edge.

In Chapter 3 we start our analysis of efficient algorithms for such a centrality measure on temporal networks by considering proxies (i.e., fast heuristics) for the temporal betweenness centrality that considers the optimal temporal paths that have minimum number of transitions [3] (i.e., shortest-temporal paths). Proxies can be more efficiently computed, and, therefore, allow for measuring the relative importance of nodes in very large temporal graphs. In this chapter, we compare several such proxies on a diverse set of real-world networks. These proxies can be divided into *global* and *local* proxies. The considered global proxies include the exact algorithm for static betweenness (computed on the underlying graph), prefix foremost temporal betweenness, which is more efficiently computable than shortest temporal betweenness, and the recently introduced approximation approach of Santoro and Sarpe [4]. As all of these global proxies are still expensive to compute on very large networks, we also turn to more efficiently computable local proxies. Here, we consider temporal versions of the ego-betweenness in the sense of Everett and Borgatti [5], standard degree notions, and a novel temporal degree notion termed the *pass-through degree*, that we introduce in this chapter and which we consider to be one of our main contributions. We show that the pass-through degree, which measures the number of pairs of neighbors of a node that are temporally connected through it, can be computed in nearly linear time for all nodes in the network and we experimentally observe that it is surprisingly competitive as a proxy for shortest-temporal betweenness.

Chapter 3 points out that there is the need for an efficient approximation algorithm for the temporal betweenness centrality that can efficiently compute high-quality approximations of the temporal betweenness scores of all the nodes and edges in temporal networks and at the same time, maintain a low computational overhead. To this end, we take a step forward in the temporal betweenness approximation by designing an efficient approximation algorithm that improves on the state-of-the-art.

In Chapter 4 we present MANTRA, a framework for approximating the temporal betweenness centrality of all nodes in a temporal graph. Our method can compute probabilistically guaranteed high-quality temporal betweenness estimates (of nodes and temporal edges) under all the feasible temporal path optimalities presented in the work

of Buß et al. [3]. We provide a sample-complexity analysis of our method and speed up the temporal betweenness computation using a state-of-the-art progressive sampling approach based on Monte Carlo Empirical Rademacher Averages. Additionally, we provide an efficient sampling algorithm to approximate the temporal diameter, average path length, and other fundamental temporal graph characteristic quantities within a small error with high probability. We support our theoretical results with an extensive experimental analysis on several real-world networks and provide empirical evidence that the MANTRA framework improves the current state of the art in speed, sample size, and required space while maintaining high accuracy of the temporal betweenness estimates.

Other examples for which real-world networks are very dynamic are peer-to-peer (P2P) in which users continuously change over time, wireless and sensor networks where mobile agents move around and move in and out of each other's transmission radius, machines that fail and must be replaced by new ones without compromising the operations that are running on the remaining network.

One of the best examples of dynamic networks among the ones mentioned above, are P2P networks in which peers (i.e., nodes) join and leave at a very high rate leading to drastic network topology changes over time. Another characteristic of these networks is that they are bandwidth-constrained, very unreliable due to the high node replacement rate, and the open admission nature (i.e., every user can easily join the network without any background check) of these systems allows *Byzantine* (i.e., malicious) nodes to join the network and try to "disrupt" the system. Due to this high level of dynamicity, performing efficient computation in distributed dynamic networks is much more challenging than in traditional static distributed systems. For starters, one has to deal with "failures" (i.e., nodes joining and leaving the network) as a "habit" rather than an exception. To continue, time and communication constraints are much more strict, thus it will be too expensive or even impossible to run a static algorithm from scratch every time that the topology changes. Moreover, this is not even an option, that is because the network can change during the new execution of the protocol and make the algorithm "fail".

We begin our journey in the distributed computing realm by empirically studying the behavior of the Bitcoin network formation protocol. The Bitcoin protocol is designed to hide the global network structure: while most of the nodes of the network can be easily discovered, the existence of an edge between two nodes is only known by the two endpoints. However, by observing the parameters of the network formation protocol, it is possible to have an overall intuition of how the network behaves (i.e., adapts to changes in the topology), and about the performances of fundamental distributed protocols such as the *flooding* protocol.

In Chapter 5 we extend a dynamic random graph model inspired by the network formation process in the Bitcoin protocol [6] that quickly converges to a *static* expander graph with high probability. We extend such a model to obtain dynamic random graph models that evolve forever: in the first model, edges can be faulty, i.e., each edge at each round disappears with some probability; in the second one, at every round new nodes join the network according to a Poisson process and each node currently in the network disappears with certain probability; in the third one, we consider a combination of the two models above, in which edges can be faulty and nodes can join and leave the network. We run extensive simulations to measure the "flooding time" in the three models, i.e., how long it takes a message starting at a random node to reach all, or almost all, the nodes. The simulations show that, for large ranges of the parameters of the models, the flooding time is short, i.e., compatible with a logarithmic growth, as a function of the number of nodes in the network. Our results also suggest that the default values of the network formation parameters used in the main implementation of the Bitcoin protocol seem overwhelmingly safe with respect to the stability of the network, and they might safely be tuned to reduce network traffic.

Another fundamental problem to address on highly dynamic networks such as P2P networks is to maintain distributed data structures in the presence of a high churn rate (nodes leaving and joining at the same time instant). To deal with more structured data in P2P networks several distributed data structures have been developed such as Skip Graphs (Aspnes and Shah [7]), SkipNets (Harvey et al., [8]), Rainbow Skip graphs (Goodrich et al., [9]), and Skip+ (Jacob et al., [10]). They 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 can be seen as a major bottleneck in the implementation and use of data structures for P2P systems. Furthermore, several works deal with the problem of the maintenance of a specific graph topology [11–14], solving the agreement problem [15], electing a leader [16], and storage and search of data [17] under adversarial churn. Unfortunately, these structures are not conducive to efficient searching and querying.

In Chapter 6 we study robust and efficient distributed algorithms for building and maintaining distributed data structures in dynamic Peer-to-Peer (P2P) networks that 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 algorithm that builds and maintains with high probability a skip list for $poly(n)$ rounds despite $\mathcal{O}(n/\log n)$ churn *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 it is oblivious to the random choices made by the algorithm). Moreover, the maintenance overhead is proportional to the churn rate. Furthermore, the algorithm is scalable since the messages are small (i.e., at most polylog(n) bits) and every node sends and receives at most polylog(n) messages per round. Our algorithm crucially relies on novel distributed and parallel algorithms to merge two n-elements 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. To the best of our knowledge, our work provides the first-known fully-distributed data structure that provably works under highly dynamic settings (i.e., high churn rate). Furthermore, they are localized (i.e., do not require any global topological knowledge). 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.

Organization of the Thesis. Chapter 2 introduces fundamental definitions and mathematical tools that are utilized throughout the thesis. The thesis is structured into two main parts: the first focuses on efficient algorithms for temporal graph mining, while the second explores distributed algorithms for dynamic networks.

In Chapter 3, we discuss efficient heuristics to approximate the temporal betweenness rankings. In Chapter 4, we introduce a novel approximation algorithm for the temporal betweenness of all nodes in a temporal network and approximating temporal distance-metrics using sampling.

In Chapter 5 we discuss several dynamic random graphs models to build resilient networks that evolve over time. While in Chapter 6, we propose an efficient distributed protocol to maintain distributed data structures on highly dynamic networks.

Finally, Chapter 7 concludes the thesis by summarizing key findings and discussing open problems and future research directions.

The Appendix provides supplementary materials, including additional experiments and theoretical results. Appendix A presents further theoretical insights on temporal betweenness approximation, along with additional experimental results omitted from Chapter 4. Appendix B includes further experimental results related to the dynamic graphs introduced in Chapter 5. Lastly, Appendix C outlines useful properties of the randomized data structure discussed in Chapter 6, along with additional theoretical findings.

Chapter 2

Background

This chapter introduces our notation and terminology commonly used throughout this thesis. Additional definitions relevant for individual chapters are introduced where needed. We proceed by formally introducing the terminology and concepts that we use in what follows. More precisely, in Section 2.1 we provide key definitions for static and temporal graphs that are useful to understand the results in Chapter 3 and Chapter 4. Furthermore, in Section 2.2 we describe the distributed computing model considered in Chapter 5 and Chapter 6. We conclude this Chapter with the description of the probabilistic tools used throughout this Thesis (see Section 2.3). Finally, we use $[k]$ with $k \in \mathbb{N}$, to denote the set $\{1, \ldots, k\}$. For a set X we denote its cardinality with $|X|$.

2.1 Graphs and Temporal Graphs

2.1.1 Static Graphs

We start by introducing standard *static*, i.e., non-temporal, graphs[1]. Throughout this section we mostly focus on *directed* graphs.

Definition 2.1. A directed (static) graph is a pair $G = (V, E)$, where $V = \{u_1, \ldots, u_n\}$ is a set whose elements are called vertices or nodes, and $E = \{(u_i, u_j) : u_i, u_j \in V \wedge i \neq j\}$ is the set of directed edges. A graph is said to be undirected if $E = \{\{u_i, u_j\} : u_i, u_j \in V \wedge i \neq j\}$.

We denote by $n = |V|$ and $m = |E|$ the number of nodes and edges in a graph, respectively. Furthermore, given a node $u \in V$ we refer to $N^{\text{in}}(u) = \{v \in V : (v, u) \in E\}$

[1] We use the terms "graph" and "network" interchangeably.

and $N^{\text{out}}(u) = \{w \in V : (u, w) \in E\}$ as the set of *in-neighbors* and of *out-neighbors* of a vertex $u \in V$, respectively. The *in-degree*, *out-degree* and *degree* of a vertex $u \in V$ are defined as $d^{\text{in}}(u) = |N^{\text{in}}(u)|$, $d^{\text{out}}(u) = |N^{\text{out}}(u)|$, and $\mathrm{d}(u) = |N^{\text{in}}(u) \cup N^{\text{out}}(u)|$, respectively. We denote by $N(u) = \{v \in V : \{u, v\} \in E\}$ the set of u's neighbors in G when the graph is *undirected*. The (undirected) degree of a vertex $u \in V$ is defined as $\mathrm{d}(u) = |N(u)|$. For a subset of nodes $U \subseteq V$, we call $G[U] := (U, E')$, where $E' := \{(u, v) \in E : u, v \in U\}$, the *induced subgraph* of U.

2.1.2 Paths and Betweenness Centrality

Metrics that enable the comparison between different networks or better understanding of a network's structure are key tools for data mining on networks. A family of important metrics are centrality measures for nodes, i.e., functions that assign to each node a score capturing its centrality in the network. Betweenness centrality [2] is one of the most famous centrality measures, and it is based on the notion of paths

Definition 2.2. Given a graph $G = (V, E)$, a path $p = \langle e_1 = (u_1, v_1), \ldots, e_k = (u_k, v_k)\rangle$ is a sequence of edges $e_i \in E$, $1 \leq i \leq k$ such that $u_i = v_{i-1}$ for $2 \leq i \leq k$.

In the above definition, path $p_{u_1 v_k}$ from u_1 to v_k has *length* $|p_{u_1 v_k}| = k$. Moreover, $p_{u_1 v_k}$ is said to be a *shortest path* from u_1 to v_k if there does not exists another path $p'_{u_1 v_k}$ (with the same endpoints) such that $|p'_{u_1 v_k}| < |p_{u_1 v_k}|$. Given a path $p_{sz} = \langle e_1 = (s, v_1), \ldots, e_k = (u_k, z)\rangle$ we say that a node $v \in V$ is *internal* to p_{sz} if v appears in p_{sz} and it is not one of the two endpoints, i.e., $v \in p_{sz}$ and $v \neq s \neq z$. Moreover, given a path p_{sz} we refer to $\mathbf{Int}(p_{sz}) = \{v \in V : v \in p_{sz} \wedge v \text{ is an internal node}\}$ as p_{sz}'s set of internal nodes. Notice that there can be multiple shortest paths between s and z and we denote the set of these paths as Γ_{sz} and the number of these paths as $\sigma_{sz} = |\Gamma_{sz}|$. If there is no path between s and z, then $\Gamma_{sz} = \{p_\emptyset\}$ where $p_\emptyset$ is an empty path[2]. Similarly to σ_{sz}, given a node $v \in V$ we denote with $\sigma_{sz}(v)$ the number of shortest paths from s to z to which v is internal to.

Definition 2.3. Given a graph $G = (V, E)$, the *normalized* betweenness centrality of a node $v \in V$ is defined as,

$$b_v = \frac{1}{n(n-1)} \sum_{s \neq v \neq z} \frac{\sigma_{sz}(v)}{\sigma_{sz}} \in [0, 1].$$

Such a statistic can be computed exactly for every node in the (unweighted) network in time $\mathcal{O}(nm)$ the network's size using Brandes's algorithm [18]. Unfortunately, this

[2]Note that even if $p = \emptyset$, the set $\{p_\emptyset\}$ is not empty. It contains one element.

algorithm quickly becomes impractical on nowadays' networks with billions of nodes and edges. Moreover, there is a theoretical evidence, in form of several conditional lower bounds results [19], for believing that a faster algorithm cannot exists, even for *approximately* computing the betweenness.

We conclude this section with the definition of the ego-betweenness centrality. Such a centrality measure (in undirected graphs) was introduced by Everett and Borgatti [5] as a more tractable variant of betweenness.

The *ego-network* $G^{[v]}$ of a node v in a static graph G is the induced subgraph of its in- and out-neighbors, i.e., $G^{[v]} := G[N^{\text{in}}(v) \cup N^{\text{out}}(v)]$. The *ego-betweenness (centrality)* of v is the betweenness of v in its ego-network, i.e.,

$$\text{ego-b}(v) := \text{b}_v(G^{[v]})$$

Everett and Borgatti [5] propose an algorithm to compute the ego-betweenness of a single node in an undirected static graph via computation of the square of the incidence matrix of the node's ego-network. We note that in the worst case the ego-network is of the same size as the original graph. For computing the temporal ego-betweennesses of all nodes, this algorithm can thus be implemented in time $O(n^{\omega+1})$, where ω is matrix multiplication exponent, i.e., the smallest real number such that two $n \times n$ matrices can be multiplied within $O(n^{\omega+\varepsilon})$ field operations for all $\varepsilon > 0$. The current best bound on ω is 2.3728596 [20].

2.1.3 Temporal Graphs

Definition 2.4. A directed *temporal graph* is an ordered tuple $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ where V is the set of nodes, $\mathcal{E} = \{(u, v, t) : u, v, \in V \wedge u \neq v \wedge t \in \mathcal{T}\}$ is the set of (directed) *temporal edges*, and T is the set of time instants[3] t in which at least one temporal edge is present in the network at time t.

We denote by $n = |V|$, $M = |\mathcal{E}|$, and $T = |\mathcal{T}|$ the number of nodes and edges in a temporal graph, and the number of unique elements in the set of time instants, respectively. Similarly for static graphs, given a subset of nodes $U \subseteq V$, we call $\mathcal{G}[U] := (U, \mathcal{E}')$, where $\mathcal{E}' := \{(u, v, t) \in \mathcal{E} : u, v \in U\}$, the *induced temporal subgraph* of U. Moreover, given a temporal graph $\mathcal{G}$, it is possible to obtain an *underlying static graph*[4]

[3]The value $\mathcal{T}$ denotes the *life-time* of the temporal graph, and, without loss of generality for our purposes, we assume that, for any $t \in \mathcal{T}$, there exists at least one temporal arc at that time and without loss of generality we assume $\mathcal{T} = [1, |\mathcal{T}|]$.

[4]From now on, we will refer to underlying static graph as underlying graphs.

Definition 2.5. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ we denote with $G_{\mathcal{G}} = (V, E)$ its underlying static graph with $E = \{(u, v) : \exists (u, v, t) \in \mathcal{E}\}$.

It is worth mentioning that the $G_{\mathcal{G}}$ is often a lossy representation of the network $\mathcal{G}$, since it ignores the timing of the events in the entire network.

2.1.4 Temporal Paths and Characteristic Quantities

Definition 2.6. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, and two nodes $s, z \in V$, a *temporal path* $tp_{sz} \subseteq V \times V \times T$ is a (unique) sequence of time-respecting temporal edges $((u_1, u_2, t_1), \dots, (u_{k-1}, u_k, t_{k-1}))$ such that for each $1 \leq i < k$, $t_i < t_{i+1}$, every node u_i is visited at most once and $u_1 = s$ and $u_k = z$.

Note that in a temporal path we are accounting for the timing of the various edges of the sequence defining the path. While we are requiring the edges on the above paths to be strictly increasing with respect to their timestamps (i.e., $t_{i+1} > t_i, 1 \leq i < k-1$), the techniques we will present can be adapted to work under non-strictly increasing case (i.e., $t_{i+1} \geq t_i, 1 \leq i < k-1$) depending on the temporal paths considered. Given a temporal path tp_{sz} from s to z, as for static graphs, we say that a node v is *internal* to tp_{sz} if it appears on a temporal edge in tp_{sz} and is different from s and z. In a temporal graph, a path from s to z can be *optimal* according to different criteria, as described next,

Definition 2.7. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and two nodes $s, z \in V$, let $tp_{sz} = ((u_1, u_2, t_1), (u_2, u_3, t_2), \dots, (u_{k-1}, u_k, t_{k-1}))$ be a temporal path from s to z, where $u_1 = s$ and $u_k = z$. We define different optimality criteria for tp_{sz} as follows:

- *Shortest* (sh): A temporal path tp_{sz} is *shortest* if there is no other temporal path tp'_{sz} from s to z such that $|tp'_{sz}| < |tp_{sz}|$, where $|tp_{sz}|$ denotes the number of transitions (hops) in the path.
- *Foremost* (fm): A temporal path tp_{sz} is *foremost* if it arrives at z at the earliest possible time, i.e., for any other temporal path tp'_{sz} with arrival time t'_z, it holds that $t_z \leq t'_z$, where t_z is the arrival time of tp_{sz} at z.
- *Fastest* (fs): A temporal path tp_{sz} is *fastest* if it minimizes the traversal time, i.e., for any other temporal path tp'_{sz}, it holds that $(t_z - t_s) \leq (t'_z - t'_s)$, where t_s and t_z are the departure and arrival times of tp_{sz}, respectively.

- *Shortest-Foremost* (sfm): A temporal path tp_{sz} is *shortest-foremost* if there is no other path tp'_{sz} that both (i) arrives at z earlier than tp_{sz} and (ii) has a minimum number of transitions.
- *Shortest-Fastest* (sfs): A temporal path tp_{sz} is *shortest-fastest* if there is no other path tp'_{sz} that both (i) has a smaller traversal time (i.e., $(t'_z - t'_s) < (t_z - t_s)$) and (ii) has a minimum number of transitions.
- *Prefix-Foremost* (pfm): A temporal path tp_{sz} is *prefix-foremost* if it is foremost and, additionally, every prefix tp_{sv} of tp_{sz} (for any intermediate node v) is also foremost.

Figure 2.1 shows an example of the first three types of temporal paths described in Definition 2.7:

- $(s \xrightarrow{1} w \xrightarrow{5} z)$ is shortest,
- $(s \xrightarrow{1} x \xrightarrow{2} y \xrightarrow{3} b \xrightarrow{4} z)$ is foremost, and
- $(s \xrightarrow{3} u \xrightarrow{4} v \xrightarrow{5} z)$ is fastest.

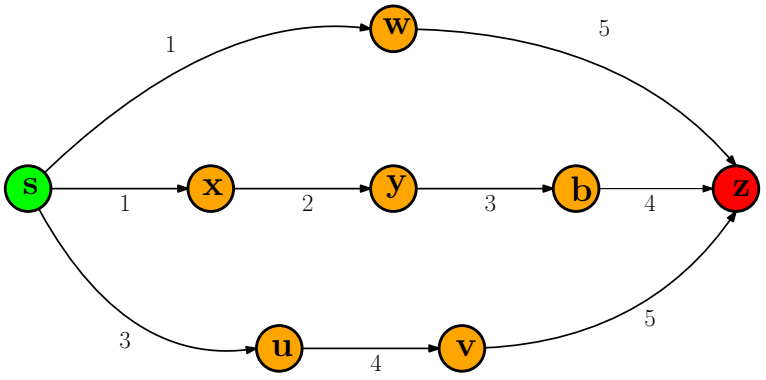


FIGURE 2.1: Example of the optimal-temporal paths: shortest, foremost and fastest described in Definition 2.7.

To denote the different type of temporal paths we use the term "$(\star)$-optimal" temporal path, where $(\star)$ denotes the type[5]. Furthermore, we denote the set of *all* $(\star)$-temporal paths between two nodes s and z as $\Gamma_{sz}^{(\star)}$ and we let

$$\mathbb{TP}_{\mathcal{G}}^{(\star)} = \bigcup_{\substack{(s,z)\in V\times V \\ s\neq z}} \Gamma_{sz}^{(\star)}$$

In this work, we will heavily rely on two temporal graphs characteristic quantities, namely the *temporal (vertex) diameter* and the *average temporal path length*. Formally,

[5]When clear from the context, will omit the term "optimal".

Definition 2.8. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, the $(\star)$-*temporal diameter* $D^{(\star)}$ and the $(\star)$-*temporal vertex diameter* $VD^{(\star)}$ are the number of temporal edges and nodes in the longest $(\star)$-optimal path in $\mathcal{G}$, i.e.,

$$D^{(\star)} = \max\left\{|tp^{(\star)}| : tp^{(\star)} \in \mathbb{TP}_{\mathcal{G}}^{(\star)}\right\} \quad \text{and} \quad VD^{(\star)} = D^{(\star)} + 1$$

respectively.

And,

Definition 2.9. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ the *average* $(\star)$-*temporal path length* $\rho^{(\star)}$ is the average number of internal nodes in a $(\star)$-*temporal path*, i.e.,

$$\rho^{(\star)} = \frac{1}{n(n-1)} \sum_{s,z \in V} |\mathbf{Int}(tp_{sz})|.$$

Next, we provide an alternative definition of the aforementioned characteristic quantities that relies on the concept of *temporal ball*. More precisely, we define the $(\star)$-*temporal ball* centered in u at time 0 of radius h as the set of nodes v that are reachable from u starting at time 0 by a $(\star)$-temporal path of length at most h. Formally,

Definition 2.10. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, the $(\star)$-*temporal ball* centered in u at time 0 of radius h is defined as

$$B(u, h) = \{v \in V : d(u, v) \leq h \wedge \mathbb{1}[u \rightsquigarrow v]\}$$

where, $d(u, v)$ is the number of hops (crossed temporal edges) needed to reach v for the first time, and $\mathbb{1}[u \rightsquigarrow v]$ is the indicator function that assumes value 1 if u can reach v via a $(\star)$-temporal path.

Now define $|B(h)| = |\{(u, v) \in V \times V : d(u, v) \leq h \wedge \mathbb{1}[u \rightsquigarrow v]\}|$ as the overall pairs of nodes that can be reached by a $(\star)$-optimal path in h steps. The $(\star)$-temporal diameter $D^{(\star)}$ of a temporal graph can be redefined in terms of $|B(h)|$ as:

$$D^{(\star)} = \min_h \left(h : \sum_u |B(u, h)| = \sum_u |B(u, h+1)|\right) \tag{2.1}$$

Alternatively, we define the $(\star)$-*temporal effective diameter* as the $\tau^{(th)}$ percentile $(\star)$-temporal path length between the nodes. We will use this quantity to provide an error bound for the approximation of $D^{(\star)}$. Let $\tau \in [0, 1]$, then

$$D_{\tau}^{(\star)} = \min_h \left(h : \frac{\sum_u |B(u, h)|}{\sum_u |B(u, D^{(\star)})|} \geq \tau\right) \tag{2.2}$$

In a similar way, we can redefine the *average* $(\star)$*-temporal path length* as

$$\rho^{(\star)} = \frac{\sum_{u\in V}\sum_{h=1}^{D^{(\star)}-1}(|B(u,h)| - |B(u,h-1)|)\cdot h}{|B(D^{(\star)})|} = \frac{\sum_{s,z\in V}\mathbb{1}[s \rightsquigarrow z]\cdot(d(s,z)-1)}{\sum_{u,v\in V}\mathbb{1}[u \rightsquigarrow v]}$$
$$= \frac{\sum_{s,z\in V}\sum_{u\in V}\mathbb{1}[u \in tp_{sz}]}{|B(D^{(\star)})|} \tag{2.3}$$

Finally, we define the $(\star)$*-temporal connectivity rate* a new measure of connectivity that allows to quantify "how well connected" a temporal graph is. The $(\star)$-temporal connectivity rate is the ratio of the number of couples that are temporally connected by a $(\star)$-temporal path and all the possible reachable couples. Formally, the temporal connectivity rate is defined as follows:

Definition 2.11. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ the $(\star)$-temporal connectivity rate is defined as

$$\zeta^{(\star)} = \frac{1}{n(n-1)}\sum_{u\neq v}\mathbb{1}[u \rightsquigarrow v] \in [0,1]. \tag{2.4}$$

Intuitively, the higher the connectivity rate the higher the number of couples that are connected via at least one $(\star)$-temporal path.

2.1.5 Temporal Betweenness Centrality

As for static networks, the importance of nodes in a network can be identified through the analysis of centrality measures. In this thesis we are particularly interested in the *temporal betweenness centrality*, that similarly to static networks, it seeks to pinpoint nodes that are traversed by a significant number of optimal (temporal) paths. Buß et al. [3, 21] gave several definitions of the *temporal betweenness* as a temporal counterpart of the *betweenness centrality*, characterized their computational complexity, and provided polynomial time algorithms to compute these temporal centrality measures. However, these algorithms turn out to be impractical, even for medium size networks. Moreover, as previously shown, on temporal graphs, there are several notions of optimal paths. Hence, we have different notions of temporal betweenness centrality as well. Formally, for any pair (s, z) of distinct nodes $(s \neq z)$, let $\sigma_{sz}^{(\star)} = |\Gamma_{sz}^{(\star)}|$ be the number of $(\star)$-temporal paths between s and z, and let $\sigma_{sz}^{(\star)}(v)$ be the number of the $(\star)$-temporal paths between s and z that *pass through* node v, with $s \neq v \neq z$. The normalized *temporal betweenness centrality* $\mathrm{b}_v^{(\star)}$ of a node $v \in V$ is defined as

Definition 2.12. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, the normalized *temporal betweenness centrality* of a node $v \in V$ is defined as

$$\mathrm{b}_v^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq v \neq z} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} \in [0, 1].$$

Computing such scores for each node in the temporal network, differently from the static case, is challenging [3]. Unfortunately, computing the temporal betweenness for some particular optimality criterion is #P-Hard, therefore we will only focus on those criteria leading to formulations that are computable in polynomial time. Finally, we extend the ego-betweenness to temporal graphs as follows. The *ego-network* $\mathcal{G}^{[v]}$ of a node v in a temporal graph $\mathcal{G}$ is the temporal graph with the underlying graph $G^{[v]} := G[N^{\mathrm{in}}(v) \cup N^{\mathrm{out}}(v)]$ and with the temporal edges being the restriction $\mathcal{E}$ to edges in $G^{[v]}$. The $(\star)$-*temporal ego-betweenness* of v is the $(\star)$-temporal betweenness of v in its temporal ego-network, i.e.,

$$\text{ego-}b_v^{(\star)} := b_v^{(\star)}(\mathcal{G}^{[v]})$$

Related Work. Nicosia et al. [22] introduced different temporal graph notions, such as temporal centralities, temporal motif, temporal clustering, temporal modularity, and temporal communities. Providing top-k algorithms for estimating temporal closeness centrality has also already been treated in the literature [23, 24]. Subsequently, a closeness variant based on bounded random-walks, related to the concept of influence spreading, has been proposed by Haddadan et al. [25]. Furthermore, Tang et al. [26] introduced temporal variants of both closeness and betweenness centrality based on foremost temporal paths, and experimentally showed the effectiveness of such metrics in spotting influential users in real-world temporal graphs. Building upon this direction, Tang et al. [27] used the notion of temporal closeness to provide an empirical analysis of the containment of malware in real-world mobile phone networks. The Katz centrality [28] has been adapted to the temporal setting [29, 30] as well, while Rozenshtein et al. [31] defined the temporal PageRank by replacing random walks with temporal random walks.

Tsalouchidou et al. [32] extended the well-known Brandes algorithm [18] to allow for distributed computation of betweenness in temporal graphs. Specifically, they studied shortest-fastest paths, considering the bi-objective of shortest length and shortest duration. Buß et al. [3] analysed the temporal betweenness centrality considering several temporal path optimality criteria, such as shortest (foremost), foremost, fastest, and prefix-foremost, along with their computational complexities. They showed that, when considering paths with increasing time labels, the foremost and fastest temporal betweenness variants are $\#P$-hard, while the shortest and shortest foremost ones can

be computed in $O(n^3T^2)$, and the prefix-foremost one in $O(nM \log M)$. Here n is the number of nodes and M the number of temporal arcs. The complexity analysis of these measures has been further refined since [21].

Santoro et al. [4] provided ONBRA, the first sampling-based approximation algorithm for one variant of the temporal betweenness centrality. The input to ONBRA is a temporal graph, a confidence value $\delta \in (0, 1)$, and the sample size r. The algorithm performs a set of r truncated temporal breadth first searches between couples of nodes sampled uniformly at random and estimates the shortest temporal betweenness using the temporal equivalent of the ABRA estimator [33] for static networks. ONBRA's output is a function of the confidence $\delta \in (0, 1)$ and the upper bound on the approximation accuracy $\xi \in (0, 1)$ computed using the *Empirical Bernstein Bound* [34]. More precisely, with probability $1 - \delta$, the approximation computed by ONBRA is guaranteed to have absolute error of at most ξ for each node in the temporal graph.

Ghanem et al. [35] defined a temporal version of ego betweenness based on most recent paths, which are paths that give the most recent information to the destination vertex about the status of the source, i.e., no other path starts from the source at a later point in time. Their definition of temporal ego betweenness is snapshot based, i.e., it gives the ego betweenness of the temporal ego graph at a specific time instant. Simard et al. [36], on the other hand, studied a continuous-time scenario of the shortest paths betweenness.

Finally, Oettershagen et al. [37] defined a random temporal walks based centrality that quantifies the importance of a node by measuring its ability to obtain and distribute information in a temporal network. They provide exact and approximate algorithms for computing their centrality measures and compare it with the state-of-the-art temporal centralities, i.e., with PageRank [31], Katz [30], closeness [23, 24], and betweenness [3].

2.2 Dynamic Networks

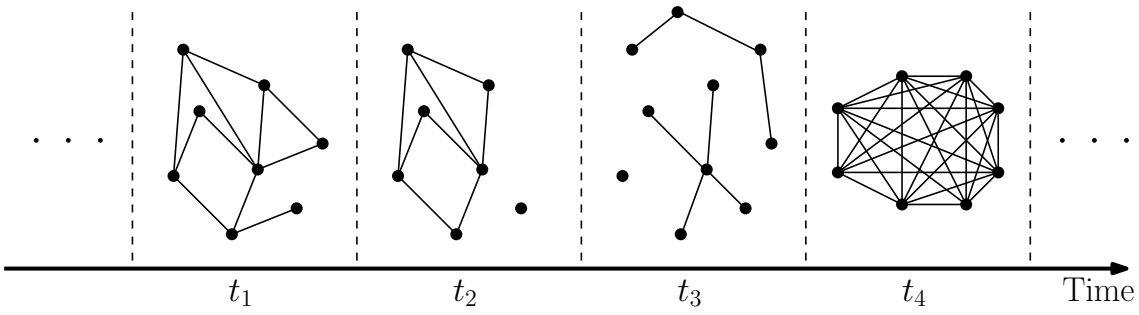


FIGURE 2.2: Example of a dynamic graph representation.

2.2.1 Dynamic Networks with Churn Model

The following model for dynamic networks that incorporates churn was first proposed in [15] and has since been used in subsequent works with suitable modifications [11, 15–17]. Henceforth, we will call the model discussed below as the DNC (Dynamic Network with Churn) model. A key parameter of the DNC model is the *churn rate*, i.e., the number of nodes that can join or leave the network. The DNC model is represented by a graph with a dynamically changing topology (both nodes and edges change from round to round) whose nodes execute a distributed algorithm and whose edges represent connectivity in the network. The focus will be on time (number of communication rounds) and number of messages – the two traditional complexity measures of distributed algorithms. A dynamic network is formally represented by a graph process $\mathcal{G} = (G_0, G_1, \ldots, G_t, \ldots)$ where $G_t = (V_t, E_t)$ in which each G_t is an *undirected* graph. In this thesis we consider two different types of churn rates, random and adversarial.

2.2.2 Random Churn Rate

We consider synchronous dynamic networks that evolve according to some probability distribution. Communication is via message passing. Nodes can send messages of size $\mathcal{O}(\log n)$ bits to each other if they know their IDs, but no more than $\mathcal{O}(\text{polylog}(n))$ incoming and outgoing messages per round. A random churn rate can be seen as a graph process $\mathcal{G} = (G_0, G_1, \ldots, G_t, \ldots)$ that generates a new graph at time t given the one at $t-1$. The churn rate follows a specific distribution, i.e., to obtain G_t from G_{t-1} a new random number of nodes or edges can be added or deleted.

2.2.3 Adversarial Churn Rate

We consider synchronous dynamic networks controlled by an *oblivious adversary*. Communication is via message passing. Nodes can send messages of size $\mathcal{O}(\log n)$ bits to each other if they know their IDs, but no more than $\mathcal{O}(\text{polylog}(n))$ incoming and outgoing messages per round. The adversary fixes a sequence of nodes $\mathcal{V} = (V_1, V_2, \ldots, V_i, \ldots)$ where $V_t \subset \mathcal{U}$, for some universe of nodes $\mathcal{U}$ and $t \geq 1$, denotes the set of nodes present in the network at round t. For the sake of simplicity in the exposition, we assume that the number of nodes $|V_t|$ is stable[6], i.e., $|V_t| \in [n, f \cdot n]$ for some fixed $f \geq 1$. Each node has a unique identifier (ID) chosen from an ID space of size polynomial in n. The edges must be viewed as a rudimentary set of connections provided by the adversary. The

[6]This assumption can be relaxed to consider a network that can shrink and grow arbitrarily.

initial knowledge graph (e.g. $G_0 = (V_0, E_0)$) can be any reasonable (sparse, low diameter, well-connected, routable, and easily constructable) structure. Moreover, for the first $B = \beta \log n$ rounds (for a sufficiently large constant $\beta > 0$) , called the *bootstrap phase*, the adversary is *silent*, i.e., there is no churn. We can think of the bootstrap phase as an initial period of stability during which the protocol prepares itself for a harsher form of dynamism.

After the bootstrap phase, the requirements on the adversary are significantly reduced. It is only required to ensure that any new node that joins the network must be connected to a distinct node or set of nodes in the network; this is to avoid too many nodes being attached to the same node, thereby causing congestion issues. Moreover, the network is said to be in its *maintenance phase* during which $\mathcal{V}$ can experience churn in the sense that a large number of nodes may join and leave dynamically at each time step.

The *churn rate* models the level of churn that the adversary can inflict on $\mathcal{V}$. It is specified by a pair (C, T) which implies that within *any* range of T consecutive rounds, at most C nodes can be added and (possibly different number of) at most C nodes can be deleted. Formally, for all $t \geq 1$ and $1 \leq i \leq T$, the adversary must ensure that $|V_{t+1} \setminus V_t| \leq C$ and $|V_t \setminus V_{t+1}| \leq C$. The churn rate we consider is $(\varepsilon \cdot n, \Theta(\log n))$ for a suitably small but fixed $\varepsilon > 0$.

Observe that the concept of adversarial churn rate *subsumes* the random one. This means that a distributed algorithm that "works" under the adversarial setting, it will properly run under the random one. However, a protocol that works under random churn rate is not guaranteed to work under the adversarial one.

2.3 Probabilistic Tools

2.3.1 Concentration Bounds

Here we list some useful concentration bounds that we will use in what follows. We will use the Hoeffding's bound in Chapter 4.

Theorem 2.13 (Hoeffding's inequality [38])**.** *Let $X_1, \ldots, X_n$ be independent bounded random variables with $a_i \leq X_i \leq b_i$, where $-\infty < a_i \leq b_i < \infty$ for all i and $\mu = \mathbf{E}\left[\sum_{i=1}^{n} \frac{x_i}{n}\right]$. Then*

$$\mathbf{Pr}\left(\left|\sum_{i=1}^{n}\frac{X_i}{n}-\mu\right|\geq\varepsilon\right)=\mathbf{Pr}\left(\left|\frac{1}{n}\sum_{i=1}^{n}\left(X_i-\mathbf{E}\left[X_i\right]\right)\right|\geq\varepsilon\right)\leq 2e^{\left(-\frac{2n^2\varepsilon^2}{\sum_{i=1}^{n}\left(b_i-a_i\right)^2}\right)} \tag{2.5}$$

for all $\varepsilon \geq 0$.

In Chapter 6, we use the Chernoff bound for the Poisson trials

Theorem 2.14 (Theorem 4.4 [39])**.** *Let* X *be a sum of* n *independent Poisson trials* X_i *such that* $\mathbf{Pr}\left(X_i=1\right)=p_i$*, for* $i\in[n]$*. Then,*

$$\mathbf{Pr}\left(X\geq R\right)\leq 2^{-R} \qquad \text{for } R\geq 6\mathbf{E}\left[X\right] \tag{2.6}$$

Moreover, to provide high confidence bounds for the properties of randomized skip list we use the following tools.

Theorem 2.15 (Right tail)**.** *Let* X *be a sum of* n *independent random variables* X_i *such that* $X_i\in[0,1]$*. Let* $\mathbf{E}\left[X\right]=\mu$*. Then,*

$$\mathbf{Pr}\left(X\geq k\right)\leq\left(\frac{\mu}{k}\right)^k\left(\frac{n-\mu}{n-k}\right)^{n-k}\leq\left(\frac{\mu}{k}\right)^k e^{k-\mu} \qquad \text{for } k>\mu \tag{2.7}$$

$$\mathbf{Pr}\left(X\geq(1+\varepsilon)\mu\right)\leq\left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right)^{\mu} \qquad \text{for } \varepsilon\geq 0 \tag{2.8}$$

Theorem 2.16 (Left tail)**.** *Let* X *be a sum of* n *independent random variables* X_i *such that* $X_i\in[0,1]$*. Let* $\mathbf{E}\left[X\right]=\mu$*. Then,*

$$\mathbf{Pr}\left(X\leq k\right)\leq\left(\frac{\mu}{k}\right)^k\left(\frac{n-\mu}{n-k}\right)^{n-k}\leq\left(\frac{\mu}{k}\right)^k e^{k-\mu} \qquad \text{for } k<\mu \tag{2.9}$$

$$\mathbf{Pr}\left(X\leq(1-\varepsilon)\mu\right)\leq e^{-\frac{\mu\varepsilon^2}{2}} \qquad \text{for } \varepsilon\in(0,1) \tag{2.10}$$

Furthermore, a useful result about the upper tail value of a negative binomial distribution to a lower tail value of a suitably defined binomial distribution that allows us to use all the results for lower tail estimates of the binomial distribution to derive upper tails estimate for negative binomial distribution. This a very nice result because finding bounds for the right tail of a negative binomial distribution directly from its definition is very difficult.

Theorem 2.17 (See Chapter 4 in [40])**.** *Let* X *be a negative binomial random variable with parameters* r *and* p*. Then,* $\mathbf{Pr}\left(X>n\right)=\mathbf{Pr}\left(Y<r\right)$ *where* Y *is a binomial random variable with parameters* n *and* p*.*

Chapter 3

Proxying Betweenness Centrality Rankings in Temporal Networks

In this chapter we present our contributions to the problem of approximating the temporal betweenness centrality rankings on very large temporal graphs. We consider proxies for the shortest temporal betweenness rankings that are more efficiently computed. Furthermore, we present a novel definition of degree of the nodes in temporal graphs that takes into account the number of length-two temporal paths *passing through* the nodes.

Throughout the chapter we use the following general approach. We employ a set of competitor algorithms that we each use as proxies for temporal betweenness rankings, i.e., for each algorithm, we compute a complete ranking of the nodes and evaluate how this ranking relates to the "correct" ranking. While different scenarios may exist, centrality values are frequently used to rank nodes and our proxy notion is motivated exactly by such applications. Formally, a proxy can be defined as follows:

Definition 3.1 (Proxy)**.** Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, two functions $f(\cdot)$ and $g(\cdot)$ that assign real valued numbers to all the nodes $u \in V$. Let $R_f(V)$ and $R_g(V)$ be the rankings of the nodes obtained after applying f and g to the temporal graph. We say that g is a good proxy for f if

$$R_g(V) \approx R_f(V)$$

Some of the considered proxies have the property that they still try to capture the global nature inherent in the definition of the shortest temporal betweenness and, as a consequence, still suffer from a comparatively bad running time, meaning that their running times are far from linear in the input size. Note however that, as argued, e.g., by Teng [41], in the age of Big Data, an algorithm should be considered efficient or scalable

if its time complexity is nearly-linear. In fact, there is even theoretical evidence, in form of several conditional lower bound results [19, 42], for believing that no such algorithm is achievable, even for *approximately* computing the betweenness values in *static* graphs. We thus shift our focus away from these *global proxies* towards *local proxies* for shortest temporal betweenness rankings. We classify a proxy as local if the centrality values of nodes are completely determined by the induced subgraph of their neighborhood (including themselves).

We perform an extensive experimental analysis of several proxies for the temporal betweenness rankings and we show that (in practice) our novel degree notion outperforms the other proxies in terms of running time and quality of the approximation.

3.1 Contribution

We compare a variety of approaches for proxying shortest temporal betweenness rankings in terms of their scalability and output quality. We start our study in Section 3.2 with a comparison of the following proxies: (1) exact algorithm for the static betweenness computed on the underlying graph, (2) the more efficiently computable prefix foremost temporal betweenness of Buß et al. and (3) the recently introduced (absolute) approximation approach of Santoro and Sarpe [4]. Our evaluation indicates that the static betweenness rankings turn out to be quite competitive, the performance of the prefix foremost temporal betweenness seems somewhat inconsistent, while the quality of the ranking returned by the considered temporal betweenness approximation algorithm very much depends on the provided time.

Next, motivated by the fact that static degree centrality is often compared to other centrality measures, we follow this approach in the temporal setting. In Section 3.3, we describe our main theoretical contribution: the *pass-through degree*, a new *temporal* degree notion which we believe to be interesting in its own right. Informally the pass-through degree of a node v measures the number of neighbor pairs of v that are temporally connected through v, i.e., that have a temporal path of length two between them that *passes through* v. We proceed by giving an algorithm that computes the pass-through degree of all nodes in a given (directed) temporal graph in $O(M \log m)$ time, where M is the number of temporal arcs and m the number of arcs in the underlying static graph. In other words, the proposed algorithm is scalable in the sense of Teng [41].

In Section 3.4 we compare the following set of local proxies in terms of their efficiency and quality: (1) temporal versions of the ego-betweenness in the sense of Everett and Borgatti [5], which entails to compute the betweenness centrality values of the nodes in

their respective ego-networks (the induced subgraph of a node's neighborhood including himself) (2) the pass-through degree, and (3) the approximation algorithm for temporal betweenness centrality also used as one of the global proxies in Section 3.2, as it is the only choice from that section that offers scalability in terms of computation time. We note that the pass-through degree falls somewhere between the simple degree notions and the ego-betweenness notion in terms of complexity. Our evaluation here indicates that the ego-networks can be of comparable size as the whole network and, thus, prohibitively large on some data sets, the pass-through degree usually does not perform worse than the ego-variants and is at the same time much faster, while the considered approximation algorithm for temporal betweenness has a more inconsistent performance over different data sets.

Our experimental evaluation is based on a diverse set of real-world networks that includes almost all publicly available networks from the works of Buß et al. [3] and Santoro and Sarpe [4]. We did not include the Karlsruhe network [43] (used in [3]) because it does not appear to be available anymore. Moreover, we replaced Mathoverflow [44] network (used in [4]) by a bigger temporal network from a different domain to make the set of analyzed temporal graphs more diverse. Finally, we excluded Ask Ubuntu and Super User [44] (also analyzed in [4]) because of the excessive amount of time needed to compute their exact temporal betweenness rankings.

3.2 Global Proxies for Shortest Temporal Betweenness

In this section, we summarize the results of our experimental study on proxying the shortest temporal betweenness values in large real-world networks using global proxies. Recall that a proxy is global, if the centrality value of each node is not purely dependent on its neighborhood. Our general experimental approach here is as follows. We employ a set of competitor algorithms that we each use as a proxy for shortest temporal betweenness centrality rankings. That is, for each algorithm, we compute a complete ranking of the nodes and evaluate (using various metrics) how this ranking relates to the "correct" ranking computed by the algorithm of Buß et al. [3]. In what follows, we will call this benchmark algorithm TempBrandes for "Temporal Brandes algorithm". Recall that TempBrandes computes the shortest temporal betweenness values of all nodes in time $O(n^3T^2)$.

3.2.1 Experimental Setting

Global Proxies. As global proxies for shortest temporal betweenness, our study includes the following algorithms.

Brandes: The classical algorithm of Brandes, which computes the static betweenness of all nodes in time $O(nm)$ on the underlying graph, i.e., the graph obtained by a union over all the time steps.[1]

Pref: The algorithm of Buß et al. [3] for computing the prefix foremost temporal betweennesses pftb in $O(nM \log M)$.

ONBRA: The approximation algorithm of Santoro and Sarpe [4], which is a sampling technique for obtaining an absolute approximation of the shortest temporal betweenness values. The work that introduced this algorithm is rather vague in terms of how to choose the sample size, stating only that they choose it so as to make the algorithm run "within a fraction of the time required by the exact algorithm". In our study, we choose the number of samples such that the running time of Onbra is a tenth, a half and roughly equal to the running time of TempBrandes. We achieve this by first estimating the time taken per sample, and then computing the number of samples by dividing the (fraction of) time needed by TempBrandes with the computed estimate.

Besides Brandes, which is available in the *Graphs.jl* library, we implemented TempBrandes and all competitor algorithms in Julia[2]. We chose to re-implement TempBrandes, Pref and Onbra because the available implementations of TempBrandes and Pref have issues with the number of paths in the tested networks, causing overflow errors (indicated by negative centralities). Since Onbra is based on the TempBrandes code, it results in the same errors. Our implementation uses a sparse matrix representation of the $n \times T$ table used in [3, 4], making the implemented algorithms space-efficient and allowing to compute the exact temporal shortest betweenness on big temporal graphs (for which the original version of the code gives out of memory errors). Furthermore, we noticed another error in TempBrandes and Pref, related to time relabeling causing an underestimation of centralities.

Networks. We evaluate all of the above competitors on real-world temporal graphs of different nature, whose properties are summarized in Table 3.1. The networks come from two different domains.

[1] We are aware of fast approximation algorithms like Kadabra [45] or SILVAN [46] for the computation of the static betweenness, but for our purpose here the efficiency of the exact algorithm is sufficient.

[2] Code available at https://github.com/Antonio-Cruciani/TSBProxy.

Data set	**n**	**m**	**M**	**T**	$\mathbf{t_{STB}}$	$\mathbf{n_e^{max}}$	**Type**	**Source**
Hypertext 2009	113	4392	41636	5246	263	99	U	[47]
High school 2011	126	3418	57078	5609	446	56	U	[47]
Hospital ward	75	2278	64848	9453	659	62	U	[47]
College msg	1899	20296	59798	58911	894	256	D	[44]
Wiki elections	7115	103680	106985	101012	1192	1066	D	[44]
High school 2012	180	4440	90094	11273	1345	57	U	[47]
Digg reply	30360	85247	86203	82641	1762	284	D	[48]
Infectious	10972	89034	831824	76944	4985	65	U	[47]
Primary school	242	16634	251546	3100	5607	135	U	[47]
Facebook wall	35817	104942	198028	194904	5751	89	D	[48]
Slashdot reply	51083	130370	139789	89862	8653	2916	D	[48]
High school 2013	327	11636	377016	7375	20642	88	U	[47]
Topology	16564	122140	198038	32823	22453	1401	U	[49]
SMS	44090	67190	544607	467838	25178	407	D	[48]
Email EU	986	24929	327336	207880	31840	346	D	[44]

TABLE 3.1: The temporal networks used in our evaluation, where n denotes the number of nodes, m the number of arcs in the underlying static graph, M the number of temporal arcs, T the number of unique time labels, t_{STB} the execution time of TEMPBRANDES, and n_e^{max} the maximum number of nodes in the ego network (type **D** stands for directed and **U** for undirected). The networks are sorted in increasing order with respect to t_{STB}.

Social networks: This domain includes most of the considered networks: College msg, Wiki elections, Digg reply, Slashdot reply, a subgraph of Facebook wall [50] containing the last $\sim 200k$ temporal arcs (as in the work of Santoro and Sarpe [4]), SMS and Email EU. These are social networks from different realms, where nodes correspond to users and temporal arcs indicate messages sent between them at specific points in time.

Contact networks: This domain includes the six networks Hypertext 2009, High school 11/12/13, Hospital ward, Infectious, Primary school and Topology. In the first case nodes correspond to individuals, while in the second case they correspond to computers. In both cases temporal arcs indicate a contact between nodes at a specific time.

Evaluation Details. We executed the experiments on a server running Ubuntu 20.04.5 LTS 112 with processors Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz and 112GB RAM. All the correlation coefficients were computed by making use of the corresponding functions available in the Python scipy.stats module [51].

3.2.2 Experimental Results

Experiment 1: Global Proxies' Correlation to TempBrandes. In our first experiment, we run both TempBrandes and all the discussed global proxies on the networks listed in Table 3.1. We then, for each of these four algorithms (TempBrandes plus three proxies), compute the resulting node ranking and evaluate the correlation of the rankings computed by the proxies with the ranking computed by TempBrandes. Here, we employ two different rank correlation measures, i.e., (1) a weighted version of Kendall's τ coefficient based on the work of Vigna [52], and (2) the number of common highest rank nodes among the first k. We also investigated the Spearman's ρ coefficient [53] and Kendall's τ coefficient [54] of the rankings, but we omit these results since they provide similar results. We, however, note that these measures indicated similar proxy performance as (1), and at the same time we find (1) more relevant, as it gives more importance to approximating the upper part of the ranking. For the weighted Kendall's τ coeffficient, we use a hyperbolic weighting scheme, as proposed by Vigna [52], that gives weights to the positions in the ranking which decay harmonically with the ranks, i.e., the weight of rank r is $1/(r+1)$. We refrain from comparing the proxies with respect to average correlation due to outliers.

Network	Execution Time (seconds)					
	TempBrandes	Brandes	Prefix	EgoPrefix	EgoSTB	PTD
Hypertext 2009	262.58	**0.01**	2.29	25.14	-	**0.01**
High school 2011	445.62	0.02	3.39	15.81	-	**0.01**
Hospital ward	659.13	**0.01**	2.01	37.97	-	**0.01**
College msg	894.44	1.12	21.58	4.83	116.53	**0.02**
Wiki elections	1192.42	6.52	49.84	45.54	586.75	**0.06**
High school 2012	1345.06	0.03	7.77	232.90	-	**0.01**
Digg reply	1762.09	123.37	224.58	1.61	4.43	**0.05**
Infectious	4985.19	3.28	50.26	26.97	820.73	**0.11**
Primary school	5607.17	0.08	39.22	492.73	-	**0.04**
Facebook wall	5750.73	349.01	429.38	2.00	17.86	**0.07**
Slashdot reply	8652.54	442.75	1116.99	7.08	38.78	**0.07**
High school 2013	20641.71	0.11	95.49	200.89	-	**0.09**
Topology	22452.98	124.98	1017.78	905.69	-	**0.08**
SMS	25178.27	129.53	591.98	4.18	476.71	**0.09**
Email EU	31839.72	0.54	180.86	411.07	-	**0.05**

Table 3.2: For each network, we show the execution times of TempBrandes and of all proxies (except for Onbra) in seconds. Dashes indicate that the experiment was interrupted after the time of TempBrandes elapsed. We omit Onbra from the table as its running time is fixed to approximately 1/10, 1/2, or 1 times the running time of TempBrandes due to the choice of the sample size.

Running Times. The running times of the global proxies can be found in the first three columns of Table 3.2. We note that Prefix always terminates in at most 15%

of the running time of TempBrandes, while Brandes always finishes in at most 7% of the running time of TempBrandes. The efficiency of both proxies is particularly pronounced on contact networks with lots of temporal edges and comparatively few edges in the underlying graph. As a result the underlying graph is comparatively small, which is beneficial for Brandes. On the other hand, the number of prefix foremost shortest paths is also much smaller than the total number of shortest temporal paths, which is beneficial for Prefix. The running times of the three Onbra versions are fixed to approximately 1/10, 1/2, and 1 times the running times of TempBrandes due to the choice of the sample size.

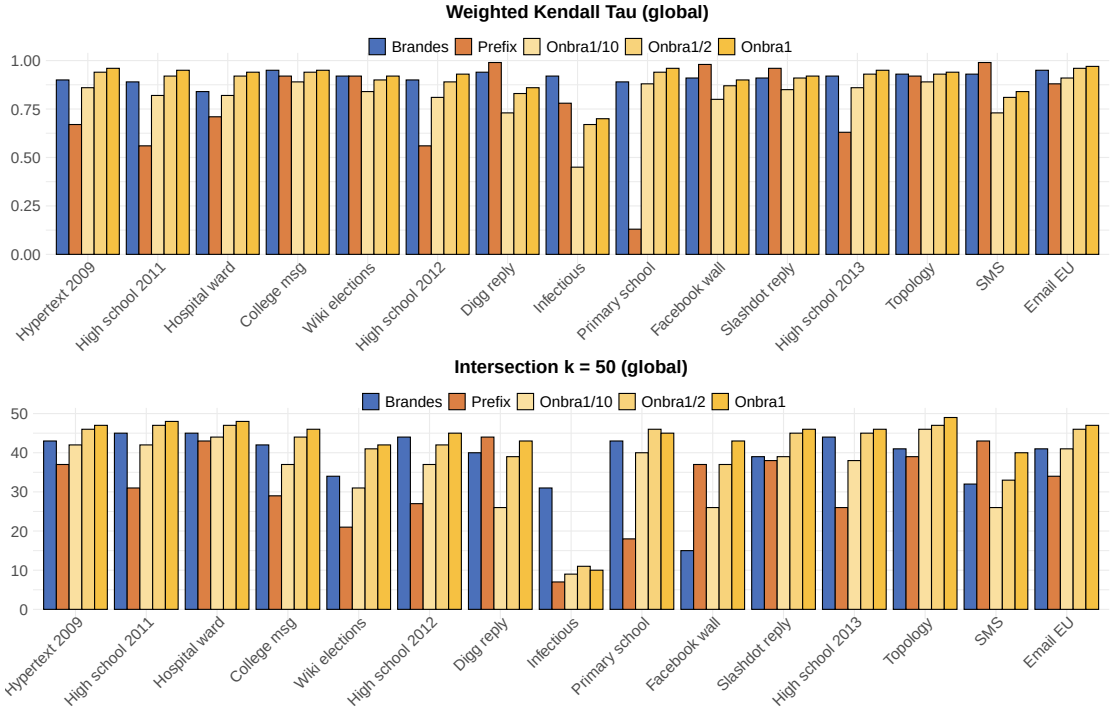


Figure 3.1: Comparison of the centrality ranking produced by TempBrandes and the rankings produced by the global proxies. The comparison is given in terms of the weighted Kendall's τ coefficient and the intersection of the top 50 nodes.

Ranking Correlation. An illustration of the ranking correlation results of this experiment can be found in Figure 3.1. On top of the figure, we show the Weighted Kendall's τ correlation of the rankings computed by the respective proxies and the ranking computed by TempBrandes. On the bottom, we show the results in terms of the intersection of the top-k nodes. We choose the value of k to be 50 here.

In terms of the weighted Kendall's τ correlation (see Table 3.3), we first observe that there are three (3) networks in which Brandes performs best, five (5) networks in which Prefix performs best, and ten (10) networks in which Onbra with maximal sample size performs best (we count networks with ties multiple times). We, however, also notice

Network	weighted Kendall's τ coefficient				
	Brandes	Prefix	Onbra $_{\frac{1}{10}}$	Onbra $_{\frac{1}{2}}$	Onbra$_1$
Hypertext 2009	0.90	0.67	0.86	0.94	**0.96**
High school 2011	0.89	0.56	0.82	0.92	**0.95**
Hospital ward	0.84	0.71	0.82	0.92	**0.94**
College msg	**0.95**	0.92	0.89	0.94	**0.95**
Wiki elections	**0.92**	**0.92**	0.84	0.90	**0.92**
High school 2012	0.90	0.56	0.81	0.89	**0.93**
Digg reply	0.94	**0.99**	0.73	0.83	0.86
Infectious	**0.92**	0.78	0.45	0.67	0.70
Primary school	0.89	0.13	0.88	0.94	**0.96**
Facebook wall	0.91	**0.98**	0.80	0.87	0.90
Slashdot reply	0.91	**0.96**	0.85	0.91	0.92
High school 2013	0.92	0.63	0.86	0.93	**0.95**
Topology	0.93	0.92	0.89	0.93	**0.94**
SMS	0.93	**0.99**	0.73	0.81	0.84
Email EU	0.95	0.88	0.91	0.96	**0.97**

Table 3.3: For each network, we show the weighted Kendall's τ coefficient of the rankings computed by the three global proxies and the ranking computed by TempBrandes. For Onbra we show the results using, respectively, a sample size such that Onbra's execution time is $1/10, 1/2$, and exactly the one of TempBrandes. For each instance, we highlight the best result in bold font.

that the Onbra's performance heavily relies on the used sample size. Indeed, if the sample size is such that Onbra needs roughly 10% of TempBrandes running time, we observe that the numbers change as follows: there are eleven (11) networks in which Brandes achieves the best correlation and there are five (5) networks in which Prefix performs best, while Onbra never performs best.

As Brandes always terminates in less than 7% of TempBrandes' running time, and in most cases much faster, we can conclude that the static betweenness rankings are actually quite competitive in situations where we are restricted in terms of running time. In other words, it seems really necessary to give Onbra a running time similar to the exact algorithm in order for it to outperform Brandes. At this point, we would like to emphasize that our choice of sample size for Onbra is inherently impractical as it requires to run the exact algorithm first. We chose this approach in order to be as fair as possible when evaluating its performance in terms of quality. Choosing its sample size based on the time of other proxies, as, e.g., Brandes, makes its performance much worse in comparison. The results based on the intersection measure are somewhat similar, with Onbra performing slightly better.

3.3 Pass-Through Degree

Motivated by the fact that the running times of the global proxies employed in the previous section all grow much faster than linearly, we now turn to local proxies, i.e., proxies which compute centrality values purely based on nodes' neighborhoods. In the case of static graphs, it is common practice to compare more involved centrality notions to the simple degree centrality. Motivated by this fact, we here introduce a new degree notion for temporal graphs, which we will evaluate as a local proxy for shortest temporal betweenness in what follows. This new degree notion is somewhat related to the ego-betweenness, but it is in fact even simpler. In the end of this section, we will show that it can be computed for all nodes in nearly linear time in the number of temporal arcs.

Static Pass-Through Degree. With the aim of a simpler exposition, we start by giving the definition of our new degree notion for static directed graphs. We first note that the two standard degree notions in directed graphs, the in-degree $\mathrm{d}^{\mathrm{in}}(u) = |N^{\mathrm{in}}(u)|$ and the out-degree $\mathrm{d}^{\mathrm{out}}(u) = |N^{\mathrm{out}}(u)|$, both fail to observe the vertex as a whole, by taking in-going and out-going arcs into account in isolation. In undirected graphs, on the other hand, the degree of a vertex also measures the number of neighbor pairs that can reach each other by passing through u, albeit normalized by the size of the neighborhood of u. In other words, $\mathrm{d}(u) = \frac{|N(u)|\cdot|N(u)|}{|N(u)|}$. This is, of course, just an overly complicated way of writing down the identity $\mathrm{d}(u) = |N(u)|$, but we use it as motivation for defining the analogous degree notion in directed graphs. We actually give two candidate definitions first, both generalizing the above equality to directed graphs, and then argue which of the two notions is more reasonable. The two variants of a directed degree notion that we propose, for a node $u \in V$, are $d_1(u) := \frac{|N^{\mathrm{in}}(u)|\cdot|N^{\mathrm{out}}(u)|}{|N^{\mathrm{in}}(u)\cup N^{\mathrm{out}}(u)|}$ and $d_2(u) := \sqrt{|N^{\mathrm{in}}(u)| \cdot |N^{\mathrm{out}}(u)|}$. When modeling an undirected graph $G = (V, E)$ as a directed graph $D = (V, A)$, by introducing two arcs (u, v) and (v, u) for every edge $\{u, v\} \in E$, we obtain, for every node u in the undirected graph, $N(u) = N^{\mathrm{in}}(u) = N^{\mathrm{out}}(v)$ and $d_1(u) = d_2(u) = d(u)$. Thus, both notions are proper generalizations of the undirected degree.

While at first sight it is not obvious which vertex degree definition is more suitable, both of them being legitimate generalizations of the undirected degree, one of the two turns out to be better suited for measuring vertex importance. As the examples in Figure 3.2 illustrate, the first candidate, d_1, has a serious drawback. More formally, when $N^{\mathrm{in}}(u) \cup N^{\mathrm{out}}(u) \in \{N^{\mathrm{in}}(u), N^{\mathrm{out}}(u)\}$, then $d_1(u) \in \{|N^{\mathrm{in}}(u)|, |N^{\mathrm{out}}(u)|\}$. This in particular means that in such a case, contrary to our initial intention, the degree of a node depends only on the in-going or the out-going arcs. Since the second candidate

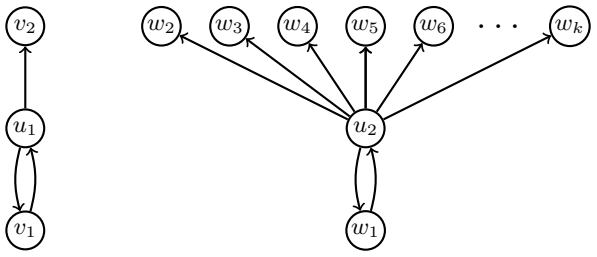


FIGURE 3.2: For the first variant, the pass-through degree of vertices u_1 and u_2 in the example graphs depicted above is equal. Namely, $d_1(u_1) = |N^{\text{in}}(u_1)| = 1 = |N^{\text{in}}(u_2)| = d_1(u_2)$. For the second variant this is not the case, as $d_2(u_1) = \sqrt{2}$ and $d_2(u_2) = \sqrt{k} = \Theta(\sqrt{n})$, where n denotes the number of nodes in the graph.

does not suffer from this issue, we find it more suitable for defining our directed degree notion. We now formally define it as the *pass-through degree* of a node.

Definition 3.2. In a static directed graph $G = (V, E)$, the *pass-through-degree* of $u \in V$ is defined as

$$\text{ptd}(u) := \sqrt{|N^{\text{in}}(u)| \cdot |N^{\text{out}}(u)|}$$

We point out that the pass-through degree is the geometric mean of in- and out-degree, the two classical notions of directed degree.

Temporal Pass-Through Degree. The introduced pass-through degree notion nicely generalizes to temporal graphs. Recall that the pass-through degree of a node u is equal to the geometric mean of the number of ordered neighbor pairs v, w that are connected through u. We generalize this to temporal nodes via pairs of neighbors that are temporally connected via exactly two hops through u. Formally, we write $v \xrightarrow{u} w$ if and only if there exist $(v, u, t_{vu}) \in \mathcal{E}$ and $(u, w, t_{uw}) \in \mathcal{E}$ such that $t_{vu} < t_{uw}$. We are now ready to define the temporal pass-through degree.

Definition 3.3. In a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, the *temporal pass-through-degree* of $u \in V$ is

$$\text{t-ptd}(u) := \sqrt{|\{(v, w) \in (V \setminus \{u\})^2 : v \xrightarrow{u} w\}|}$$

Computation of the Temporal Pass-Through Degree. Algorithm 1, given the temporal arc list $\mathcal{E}$ of a temporal graph, computes the pass-through degrees in $O(M \log m) = \tilde{O}(M)$ time and $O(m+n)$ space, where M is the number of temporal arcs and m, n are, respectively, the number of arcs and the number of nodes of the underlying static graph. More precisely, the first **for** loop (lines 2-7) iterates over all the temporal arcs and builds two simple labeled directed graphs, $\overline{G}$ and $\underline{G}$, which respectively keep track of the maximum and the minimum appearance time of each arc from the underlying graph. Building $\overline{G}$ and $\underline{G}$ requires $O(M \log m)$ time, as we can maintain a vertex-sorted list of already

Algorithm 1: Temporal Pass-Through Degree

Data: temporal edge list $\mathcal{E}$
Result: temporal pass-through degree of all vertices t-ptd

1 $\overline{G}, \underline{G} = \{\emptyset\}$ // initialize two empty temporal graphs
2 **for** ***each*** $(u, v, t) \in \mathcal{E}$ **do**
  // check if the edge already exists in $\overline{G}, \underline{G}$
3 **if** $(u, v) \in E(\overline{G})$ **then**
    // update max and min encountered label
4 $\overline{G}(u, v) = \max\left(\overline{G}(u, v), t\right)$, $\underline{G}(u, v) = \min\left(\underline{G}(u, v), t\right)$
5 **else**
6 add (u, v) to $E(\overline{G}), E(\underline{G})$
7 $\overline{G}(u, v) = t$, $\underline{G}(u, v) = t$
8 sort edges of $\underline{G}$ in ascending order according to time labels
9 $L_{eat} = [[\cdot], [\cdot], \ldots, [\cdot]]$ // list of n empty lists
10 **for** ***each*** $(u, v, \underline{t}) \in \underline{G}$ **do**
11 $L_{eat}[v].append(\underline{t})$
12 t-ptd $= [0, 0, \ldots, 0]$// initialize array of n zeros
13 **for** ***each*** $(v, w, \overline{t}) \in \overline{G}$ **do**
  // compute the pass-through degrees
14 t-ptd$[v]$ = t-ptd$[v] + \left|\{t \in L_{eat}[v] : t < \overline{t}\}\right|$
15 **return** t-ptd

added arcs, and $O(m + n)$ space. Subsequently, the algorithm sorts the m arcs of the temporal graph $\underline{G}$ according to their time labels in time $O(m \log m)$. The second **for** loop (lines 10-11) iterates over all the (now sorted) arcs of the temporal graph $\underline{G}$, and appends the appearance time $\underline{t}$ of the arc $(u, v, \underline{t})$ to the minimum incoming times list of node v. Since $\underline{G}$ has exactly m arcs, this loop requires $O(m)$ steps and uses $O(m + n)$ space. Finally, using $O(n)$ space, the last **for** loop (lines 13-14) iterates over all the m temporal arcs $(u, w, \overline{t})$ of $\overline{G}$ and increments the t-ptd variables. More specifically, when encountering the temporal arc $(v, w, \overline{t})$, it increases the previous t-ptd value of v by the number of distinct in-going temporal arcs of v in $\underline{G}$ with $t < \overline{t}$ (line 14). Since $L_{eat}[v]$ is a sorted lists of length at most n (as $\underline{G}$ is a simple graph), for each $v \in V$ we can compute the new $ptd[u]$ in $O(\log n)$ via binary-search. Therefore, the last loop requires $O(m \log n)$ steps. The overall time and space complexities are therefore $O(M \log m) = \tilde{O}(M)$ and $O(m + n)$, respectively.

3.4 Local Proxies for Shortest Temporal Betweenness

We now turn to an experimental analysis of local proxies for shortest temporal betweenness. Our approach here is the same as in Section 3.2 and, besides the different choice

of proxies, our experimental setting is identical. We first list the set of local proxies for shortest temporal betweenness that our study includes.

EgoSTB: The algorithm for computing the ego-shortest temporal betweenness ego-stb of all nodes by going through them iteratively, computing the ego-network of the respective node, and then calling the algorithm of Buß et al. [3] for computing the shortest temporal betweenness of the node in its ego-network.

EgoPrefix: The algorithm that, analogously to the one above, computes the ego-prefix foremost temporal betweenness ego-pftb of all nodes.

PTD: The algorithm for computing the temporal pass-through degree of all nodes in nearly linear time in the number of temporal arcs, described in Section 3.3.

We in addition examine the rankings produced by both the static and temporal versions of the in- and out-degree (see Table 3.4). The quality of the rankings returned by PTD is usually much better, and only in a single case (on Infectious) is the obtained Weighted Kendall's τ value more than 0.01 worse than for another degree notion.

Network	weighted Kendall's τ coefficient				
	PTD	in-degree	out-degree	t-in-degree	t-out-degree
Hypertext 2009	**0.89**	**0.89**	**0.89**	0.72	0.72
High school 2011	0.76	**0.77**	**0.77**	0.40	0.40
Hospital ward	0.82	0.83	0.83	**0.85**	**0.85**
College msg	**0.95**	0.91	0.92	0.90	0.91
Wiki el's	**0.94**	0.74	0.72	0.72	0.72
High school 2012	0.81	**0.82**	**0.82**	0.50	0.50
Digg reply	**0.96**	0.84	0.83	0.84	0.83
Infectious	0.65	**0.70**	**0.70**	0.42	0.42
Faceb'k w'l	**0.93**	0.86	0.89	0.85	0.88
Primary school	0.83	**0.84**	**0.84**	0.70	0.70
Slashdot reply	**0.96**	0.78	0.94	0.80	0.94
SMS	**0.94**	0.84	0.88	0.69	0.81
High school 2013	0.83	**0.84**	**0.84**	0.50	0.50
Topology	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**
Email EU	**0.91**	0.87	0.90	0.77	0.83

Table 3.4: For each network, we show the weighted Kendall's τ coefficient of the rankings computed by the static/temporal degree notions and the pass-through degree and the ranking computed by TempBrandes.

Experiment 2: Local Proxies' Correlation to TempBrandes.

Network	weighted Kendall's τ coefficient			
	Onbra $_{\frac{1}{10}}$	EgoPrefix	EgoSTB	PTD
Hypertext 2009	0.86	0.73	-	**0.89**
High school 2011	**0.82**	0.69	-	0.76
Hospital ward	**0.82**	0.77	-	**0.82**
College msg	0.89	0.94	0.94	**0.95**
Wiki elections	0.84	**0.94**	**0.94**	**0.94**
High school 2012	**0.81**	**0.81**	-	**0.81**
Digg reply	0.73	**0.96**	**0.96**	**0.96**
Infectious	0.45	0.76	**0.81**	0.65
Primary school	**0.88**	0.63	-	0.83
Facebook wall	0.8	**0.94**	**0.94**	0.93
Slashdot reply	0.85	**0.97**	**0.97**	0.96
High school 2013	**0.86**	0.83	-	0.83
Topology	0.89	**0.92**	-	**0.92**
SMS	0.73	0.95	**0.96**	0.94
Email EU	**0.91**	**0.91**	-	**0.91**

Table 3.5: For each network, we show the weighted Kendall's τ coefficient of the rankings computed by the three local proxies and the ranking computed by TempBrandes. For Onbra we show the results using a sample size such that Onbra's execution time is 1/10 the one of TempBrandes. For each instance, we highlight the best result in bold font.

Running Times. The local proxies' running times can be found in the last three columns of Table 3.2. We note that the running time of EgoSTB easily becomes prohibitively large: in fact, we interrupted its execution once the time of TempBrandes elapsed, which resulted in eight (8) missing values for EgoSTB. We note that this is due to the large size of the ego networks, which can be deduced from the $n_e^{\max}$ column in Table 3.1. We emphasize that the nearly linear time algorithm from the previous section computes the pass-through degree of all nodes in less than 0.005% of the running time of TempBrandes on all data sets.

Ranking Correlation. An illustration of the ranking correlation results of this experiment can be found in Figure 3.3. On top of the figure, we show the Weighted Kendall's τ correlation coefficient of the rankings computed by the respective proxies and the ranking computed by TempBrandes (see also Table 3.5). On the bottom, we show the results in terms of the intersection of the top-k nodes (again $k = 50$). In order to allow for better comparison with the results for global proxies from Section 3.2, in all the tables and plots that follow, we also include the fastest variant of Onbra, i.e, the variant with roughly 10% of TempBrandes' running time. We observe that the pass-through degree usually does not perform worse than the ego-variants of the shortest

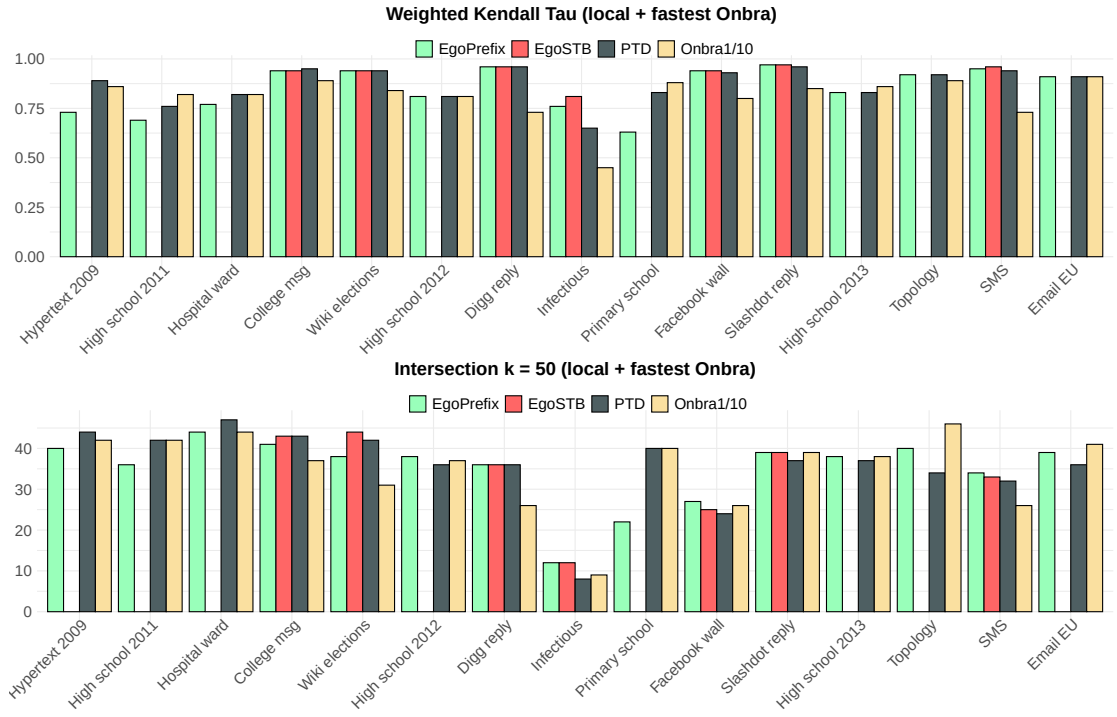


FIGURE 3.3: Comparison of the centrality ranking produced by TEMPBRANDES and the rankings produced by the local proxies and ONBRA with the smallest considered sample size. The comparison is given in terms of the weighted Kendall's τ coefficient and the intersection of the top 50 nodes.

temporal betweenness and is at the same time much faster. In terms of both weighted Kendall's τ coefficient and the intersection measure, the pass-through degree performs better or at least as good as the considered version of ONBRA on 10 out of 15 instances. At the same time its running time is between 3 and 4 orders of magnitudes smaller on all instances.

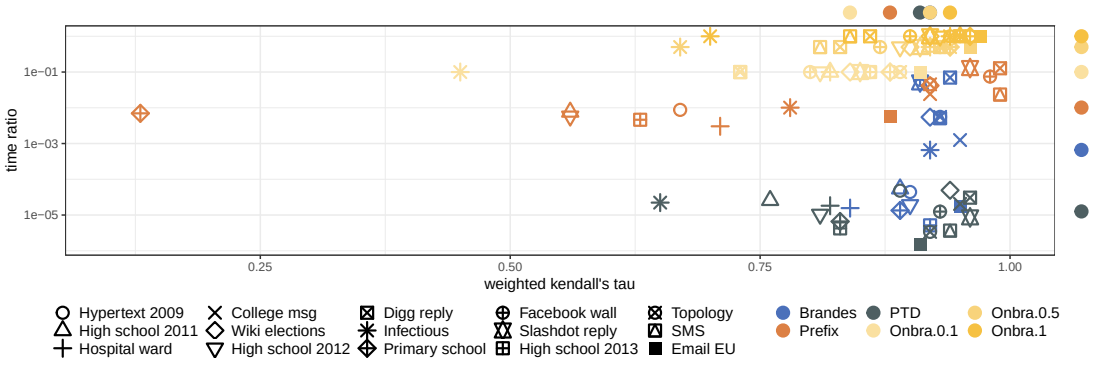


FIGURE 3.4: A two-dimensional illustration of ranking quality in terms of weighted Kendall's τ coefficient (on the horizontal linear axis) and the ratio between the proxies execution time and the time of TEMPBRANDES (on the vertical logarithmic axis). The shapes of the points indicate the network, while the color indicates the proxy. On the top and on the right we plot the median value of the weighted Kendall's τ and the time ratio, respectively. We note that the running time ratios of the three ONBRA variants are fixed to 1/10, 1/2, and 1, respectively.

Chapter 4

Approximating the Temporal Betweenness Centrality through Sampling

In this chapter we present an approximation algorithm for obtaining high-quality, probabilistically guaranteed estimates of the temporal betweenness centrality values of all the nodes in a temporal network. We then evaluate with an extensive experimental analysis on several real-world networks and provide empirical evidence that our proposed algorithm "MANTRA" improves the current state of the art in speed, sample size, and required space while maintaining high accuracy of the temporal betweenness estimates.

4.1 Contribution

We propose MANTRA (*teMporAl betweeNnes cenTrality thRough sAmpling*), a rigorous framework for the approximation of the temporal betweenness of all the vertices and temporal edges in large temporal graphs. In particular, we present the following results:

(1) We extend the state-of-the-art estimator [4] to all the feasible temporal betweenness centrality variants described in [3].

(2) We derive new bounds on the sufficient number of samples to approximate the temporal betweenness centrality for all nodes[1], that are governed by three key quantities of the temporal graph, such as the *temporal vertex diameter*, *average temporal path length*, and the *maximum variance* of the temporal betweenness

[1]The sample complexity analysis holds also for the temporal edges.

centrality estimators. Moreover, this result solves an open problem in [46, 55] on whether the sample complexity bounds for the static betweenness can be *efficiently* extended to temporal graphs. As a consequence, it significantly improves on the state-of-the-art results for the temporal betweenness estimation process [4]. Additionally, our analysis of sample complexity presents further challenges regarding the efficient computation of the three quantities upon which the bounds for the necessary sample size depend.

(3) We propose a novel algorithm to efficiently estimate the key quantities of interests in (2) that uses a mixed approach based on *sampling* and *counting*. The time complexity of our approach is $\tilde{\mathcal{O}}(\frac{\log n}{\varepsilon^2}M)$, while the space complexity is $\mathcal{O}(n+M)$. We provide an estimate on the sample size needed to achieve good estimates up to a small error bound. More precisely, we prove that $r = \Theta(\frac{\log n}{\varepsilon^2})$ sample nodes are sufficient to estimate, with probability at least $1-1/n^2$: (i) the temporal (effective) diameter $D^{(\star)}$ with error bounded by $\frac{\varepsilon}{\zeta^{(\star)}}$; (ii) the average temporal path length $\rho^{(\star)}$ with error bounded by $\varepsilon\frac{D^{(\star)}}{\zeta^{(\star)}}$; and, (iii) the temporal connectivity rate $\zeta^{(\star)}$.

(4) We define MANTRA, a progressive sampling algorithm that uses an advanced tool from statistical learning theory, namely *Monte Carlo Empirical Rademacher Averages* [56] and the above results (e.g. (1-3)) to provide a high quality approximation of the temporal betweenness. MANTRA's output is a function of two parameters: $\varepsilon \in (0,1)$ controlling the approximation's accuracy, and $\delta \in (0,1)$ controlling the confidence of the computed approximation. Our novel approach improves on ONBRA [4] (i.e., the state-of-the-art algorithm) in terms of running time, sample size, and allocated space.

(5) We support our theoretical analysis with an extensive experimental evaluation, in which we compare MANTRA with ONBRA.

4.2 Preliminaries

Here we introduce additional background needed only for this chapter.

4.2.1 Temporal Graphs, and Paths.

A directed *temporal graph* is an ordered tuple $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ where V is the set of nodes, $\mathcal{E} = \{(u,v,t) : u,v, \in V \wedge u \neq v \wedge t \in \mathcal{T}\}$ is the set of (directed) *temporal edges*, an T is the set of time instants[2] t in which at least one temporal edge is present in the network at

[2]Recall that we assume $\mathcal{T} = [|\mathcal{T}|]$, i.e., we assume that the time instants are mapped to the first $|\mathcal{T}|$ integers.

time t. Given two *distinct* nodes s and z, a *temporal path* tp_{sz} from s to z can be described as a time-ordered sequence of *vertex appearances* $tp_{sz} = ((u_1, t_1), (u_2, t_2), \ldots, (u_k, t_k))$ such that $u_1 = s$, and $u_k = z$. The vertex appearances (u_1, t_1) and (u_k, t_k) are called *endpoints* of tp_{sz} and the temporal nodes in $\mathbf{Int}(tp_{sz}) = tp_{sz} \setminus \{(u_1, t_1), (u_k, t_k)\}$ are called *internal vertex appearances* of tp_{sz}. Unlike paths on static graphs, in the temporal setting there are several concepts of optimal paths (e.g., *shortest*, *foremost*, *fastest* [3, 21, 57], see Definition 2.7). Moreover, as for the static betweenness, the task of computing the desired centrality measure boils down to the ability of efficiently *counting* the overall number of optimal paths. Unfortunately, it has been already shown that such task turns out to be #P-Hard for some temporal path optimalities (e.g. foremost, fastest) [3, 21]. Hope is left for the shortest (and all its variants) and the prefix foremost temporal paths.

We formally describe those that can be efficiently counted.

Definition 4.1. Given a temporal graph $\mathcal{G}$, and two nodes $s, z \in V$. Let tp_{sz} be a temporal path from s to z, then tp_{sz} is said to be:

(1) *Shortest* (sh) if there is no tp'_{sz} such that $|tp'_{sz}| < |tp_{sz}|$;

(2) *Shortest-Foremost* (sfm) if there is no tp'_{sz} that has an earlier arrival time in z than tp_{sz} and has minimum length in terms of number of hops from s to z;

(3) *Prefix-Foremost* (pfm) if tp_{sz} is foremost and every prefix tp_{sv} of tp_{sz} is foremost as well.

To denote the different type of temporal paths we use the same notation of Buß et al. [3]. More precisely, we use the term "$(\star)$-optimal" temporal path, where $(\star)$ denotes the type. Furthermore, we denote the set of *all* $(\star)$-temporal paths between two nodes s and z as $\Gamma_{sz}^{(\star)}$ and we let $\mathbb{TP}_{\mathcal{G}}^{(\star)}$ to be the union of all the $\Gamma_{sz}^{(\star)}$'s, for all pairs $(s, z) \in V \times V$ of distinct nodes. In this chapter, we will heavily rely on two temporal graphs characteristic quantities, namely the *temporal (vertex) diameter* $VD^{(\star)}$ and the *average temporal path length* $\rho^{(\star)}$ (see Chapter 2 for the formal definitions)[3].

4.2.2 Temporal Betweenness Centrality.

As previously shown, on temporal graphs, there are several notions of optimal paths. Hence, we have different notions of *temporal betweenness centrality* [3] as well. Formally, let $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ be a temporal graph. For any pair (s, z) of distinct nodes $(s \neq z)$, let $\sigma_{sz}^{(\star)}$ be the number of $(\star)$-temporal paths between s and z, and let $\sigma_{sz}^{(\star)}(v)$ be the number

[3]Recall that the $(\star)$-temporal vertex diameter is defined as $VD^{(\star)} = D^{(\star)} + 1$

of the $(\star)$-temporal paths between s and z that *pass through* node v, with $s \neq v \neq z$. The normalized *temporal betweenness centrality* $\mathrm{b}_v^{(\star)}$ of a node $v \in V$ is defined as

$$\mathrm{b}_v^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq v \neq z} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}}$$

We refer to Appendix A for the definition of the $(\star)$-temporal betweenness of the temporal edges. Whenever we use the term $(\star)$-temporal paths we consider $(\star)$ to be one of the optimality criteria in Definition 4.1. We observe that the average $(\star)$-*temporal path length* is equal to the sum of the $(\star)$-temporal betweenness centrality over all nodes $v \in V$.

Lemma 4.2. $\rho^{(\star)} = \sum_{v \in V} b_v^{(\star)}$

Proof. We can write the $(\star)$-temporal betweenness as

$$\mathrm{b}_v^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq v \neq z} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{tp \in \Gamma_{sz}^{(\star)}} \frac{\mathbb{1}[v \in \mathbf{Int}(tp)]}{\sigma_{sz}^{(\star)}}$$

summing over all the nodes, we obtain

$$\begin{aligned}
&\sum_{v \in V} \mathrm{b}_v^{(\star)} = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{tp \in \Gamma_{sz}^{(\star)}} \frac{1}{\sigma_{sz}^{(\star)}} \sum_{v \in V} \mathbb{1}[v \in \mathbf{Int}(tp)] \\
&= \frac{1}{n(n-1)} \sum_{s,z \in V} \frac{\sigma_{sz}^{(\star)}}{\sigma_{sz}^{(\star)}} \sum_{v \in V} \mathbb{1}[v \in \mathbf{Int}(tp_{sz})] = \frac{1}{n(n-1)} \sum_{s,z \in V} |\mathbf{Int}(tp_{sz})| = \rho^{(\star)}
\end{aligned}$$

□

4.2.3 Supremum Deviation and Empirical Rademacher Averages

Given a set of real-valued functions $\mathcal{F}$ from a domain $\mathcal{D}$ to the interval $[a, b] \subset \mathbb{R}$. We denote a sample $\mathcal{S}$ from a probability distribution π where each $s \in \mathcal{S}$ is sampled i.i.d. from π. The *empirical average* of a function $f \in \mathcal{F}$ over a sample $\mathcal{S}$ is

$$\boldsymbol{a}_f(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} f(s_i)$$

Moreover, given a random sample $\mathcal{S}$ from a distribution π, we define the *expected value* of $f \in \mathcal{F}$ taken with respect to $\mathcal{S}$ as:

$$\mathbf{E}_{\mathcal{S} \sim \pi}[f] = \mathbf{E}_{\mathcal{S} \sim \pi}[\boldsymbol{a}_f(\mathcal{S})]$$

that can be seen as the "true" value of $\boldsymbol{a}_f(\mathcal{S})$ according to the distribution π. A central question in statistics is how well $\boldsymbol{a}_f(\mathcal{S})$ *approximates* its expectation $\mathbf{E}_{\mathcal{S}\sim\pi}[f]$. The Central Limit Theorem (CLT) provides an asymptotic result that shows $\boldsymbol{a}_f(\mathcal{S})$ converging to its expected value when $|\mathcal{S}|$ goes to infinity. In this chapter we are interested in bounding the *rate of the convergence* with finite-sample bounds. In order to obtain such estimates, we use the concept of *supremum deviation* (SD) $\mathrm{SD}(\mathcal{F},\mathcal{S})$ as the maximum difference in absolute value between $\boldsymbol{a}_f(\mathcal{S})$ and its expected value over all $f \in \mathcal{F}$. Formally,

Definition 4.3. Let $\mathcal{F}$ be a set of real-valued functions from a domain $\mathcal{D}$ to the interval $[a,b] \subset \mathbb{R}$. Consider a sample $\mathcal{S}$ sampled from $\mathcal{D}$ according to a probability distribution π. Then the supremum deviation is defined as

$$\mathrm{SD}(\mathcal{F},\mathcal{S}) = \sup_{f\in\mathcal{F}} \left| \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} f(s_i) - \mathbf{E}_{\mathcal{S}\sim\pi}[f] \right|$$

The SD is the key concept in the study of empirical processes [58]. Sample-dependent quantities, such as the popular *Empirical Rademacher average* (ERA) [59] can be used to derive probabilistic upper bounds to the SD.

Definition 4.4. Let $\mathcal{F},\mathcal{D}$ and $\mathcal{S}$ defined as before. Moreover, let $\boldsymbol{\lambda} \in \{-1,+1\}^{|\mathcal{S}|}$ be a vector of i.i.d. Rademacher random variables, then the Empirical Rademacher average (ERA) of $\mathcal{F}$ on $\mathcal{S}$ is

$$R(\mathcal{F},\mathcal{S}) = \mathbf{E}_{\boldsymbol{\lambda}} \left[\sup_{f\in\mathcal{F}} \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \lambda_i f(s_i) \right]$$

Computing the ERA $R(\mathcal{F},\mathcal{S})$ is usually intractable, since there are $2^{|\mathcal{S}|}$ possible assignments for $\boldsymbol{\lambda}$ and for each such assignment a supremum over $\mathcal{F}$ must be computed. Here, we use a sharp probabilistic upper bound on the ERA that uses Monte-Carlo estimation [56].

Definition 4.5. Let $\mathcal{F},\mathcal{D},\mathcal{S}$ defined as before and let $\boldsymbol{\lambda} \in \{-1,+1\}^{c\times|\mathcal{S}|}$ for $c \geq 1$, be a $c \times |\mathcal{S}|$ matrix of i.i.d. Rademacher random variables. Then the c-Monte-Carlo Empirical Rademacher average (c-MCERA) of $\mathcal{F}$ on $\mathcal{S}$ using $\boldsymbol{\lambda}$ is

$$R^c_{|\mathcal{S}|}(\mathcal{F},\mathcal{S},\boldsymbol{\lambda}) = \frac{1}{c} \sum_{j=1}^{c} \sup_{f\in\mathcal{F}} \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \lambda_{j,i} f(s_i)$$

The c-MCERA allows to obtain sharp data-dependent probabilistic upper bounds on the SD, as they directly estimate the expected SD of sets of functions by taking into

account their correlation. Moreover, they are often significantly more accurate than other methods [46, 55, 60], such as the ones based on loose deterministic upper bounds to ERA [33], distribution-free notions of complexity such as the Hoeffding's bound or the VC-Dimension, or other results on the variance [4, 34]. Furthermore, a key quantity governing the accuracy of the c-MCERA is the *empirical wimpy variance* [61] $\mathcal{W}_{\mathcal{F}}(\mathcal{S})$, that for a sample of size r is defined as

$$\mathcal{W}_{\mathcal{F}}(\mathcal{S}) = \sup_{f \in \mathcal{F}} \frac{1}{r} \sum_{i=1}^{r} (f(s_i))^2$$

Theorem 4.6 (Theorem 4.1 in [46])**.** *For $c, r \geq 1$, let $\boldsymbol{\lambda} \in \{-1, +1\}^{c \times r}$ be a $c \times r$ matrix of Rademacher random variables, such that $\lambda_{j,i} \in \{-1, +1\}$ independently and with equal probability. Then, with probability at least $1 - \delta$ over $\boldsymbol{\lambda}$, it holds*

$$R(\mathcal{F}, \mathcal{S}) \leq R_r^c(\mathcal{F}, \mathcal{S}, \boldsymbol{\lambda}) + \sqrt{\frac{4\mathcal{W}_{\mathcal{F}}(\mathcal{S}) \ln(1/\delta)}{cr}} \tag{4.1}$$

We are ready to state the technical result of this section

Theorem 4.7. *Let $\mathcal{F}$ be a family of functions with codomain in $[0, 1]$, and let $\mathcal{S}$ be a sample of r random samples from a distribution π. Denote $\hat{v}$ such that $\sup_{f \in \mathcal{F}} Var(f) \leq \hat{v}$. For any $\delta \in (0, 1)$, define*

$$\tilde{R} = R_r^c(\mathcal{F}, \mathcal{S}, \boldsymbol{\lambda}) + \sqrt{\frac{4\mathcal{W}_{\mathcal{F}}(\mathcal{S}) \ln(4/\delta)}{cr}} \tag{4.2}$$

$$R = \tilde{R} + \frac{\ln(4/\delta)}{r} + \sqrt{\left(\frac{\ln(4/\delta)}{r}\right)^2 + \frac{2\ln(4/\delta)\tilde{R}}{r}}$$

$$\xi = 2R + \sqrt{\frac{2\ln(4/\delta)\,(\hat{v} + 4R)}{r}} + \frac{\ln(4/\delta)}{3r} \tag{4.3}$$

With probability at least $1 - \delta$ over the choice of $\mathcal{S}$ and $\boldsymbol{\lambda}$, it holds $SD(\mathcal{F}, \mathcal{S}) \leq \xi$.

The proof of this theorem follows the one by Pellegrina and Vandin in [46], for completeness we show the adapted proof. Moreover, we make use of the following theorem to prove Theorem 4.7.

Theorem 4.8 ([61])**.** *With probability at least $1 - \delta$ over $\mathcal{S}$, it holds*

$$R(\mathcal{F}, r) \leq R(\mathcal{F}, \mathcal{S}) + \sqrt{\left(\frac{\ln(1/\delta)}{r}\right) + \frac{2\ln(1/\delta)R(\mathcal{F}, \mathcal{S})}{r}} + \frac{\ln(1/\delta)}{r}$$

Proof. (Of Theorem 4.7) Define the following four events:

$$E_1 = \text{“}R(\mathcal{F},\mathcal{S}) > \tilde{R}\text{”}, \quad E_2 = \text{“}R(\mathcal{F},s) > \tilde{R}\text{”}, \quad E_3 = \text{“}\sup_{f\in\mathcal{F}}\{\mu_{\mathcal{S}}(f) - \mu_{\pi}(f)\} > \xi\text{”},$$

and $\quad E = \text{“}SD(\mathcal{F},\mathcal{S}) > \xi\text{”}.$

By applying the union bound on these events we obtain:

$$\mathbf{Pr}\,(E) \leq E_1 + E_2 + E_3$$

Now we can upper bound the probabilities of the single events by applying the *symmetrization lemma* (Theorem 14.20 in [39]), Theorem 2.3 in [62] and Theorem 4.8 replacing $\delta/4$ in the equations: $\mathbf{Pr}\,(E_1) \leq \delta/4$ follows from Theorem 4.6 by replacing δ with $\delta/4$; $\mathbf{Pr}\,(E_2) \leq \delta/4$ follows from Theorem 4.8; and, $\mathbf{Pr}\,(E_3) \leq \delta/4$ follows after using the symmetrization lemma and twice Theorem 2.3 in [62]. Moreover, the event E is true with probability at most δ. □

4.2.4 Sample Complexity and Vapnik Chervonenkis dimension

A *range space* is a pair $(\mathcal{D},\mathcal{R})$ where $\mathcal{D}$ is a (finite or infinite) domain and $\mathcal{R}$ is a (finite or infinite) family of subsets of $\mathcal{D}$. The members of $\mathcal{D}$ are called *points* and those of $\mathcal{R}$ are called *ranges*. The Vapnik-Chervonenkis (VC) Dimension of $(\mathcal{D},\mathcal{R})$ is a measure of the *complexity* of *expressiveness* of $\mathcal{R}$ [63]. Having an upper bound to the VC-dimension of a range space can be very useful: if we define a probability distribution π over $\mathcal{D}$, then a finite upper bound to the VC-dimension of $(\mathcal{D},\mathcal{R})$ implies a bound to the number of random samples from π required to approximate the probability $\pi(\mathcal{D}) = \sum_{x\in\mathcal{R}}\pi(x)$ of each range $\mathcal{R}$ simultaneously using the empirical average of $\mathcal{R}$ as an estimator.

Given a $H \subseteq \mathcal{D}$, the *projection* of $\mathcal{R}$ on $\mathcal{H}$ is defined as $\mathcal{P}_{\mathcal{R}}(H) = \{R \cap H : R \in \mathcal{R}\}$. If $|\mathcal{P}_{\mathcal{R}}(H)| = 2^{|H|}$, then H is said to be *shattered by* $\mathcal{R}$. The VC-dimension of a range space is the size of the largest subset H that can be shattered by $\mathcal{R}$, formally:

Definition 4.9. The VC-dimension of a range space $(\mathcal{D},\mathcal{R})$ denoted by $\mathrm{VC}(\mathcal{R})$ is

$$\mathrm{VC}(\mathcal{R}) = \max\left\{d : \exists H \subseteq \mathcal{D} \text{ such that } |H| = d \wedge |\mathcal{P}_{\mathcal{R}}(H)| = 2^d\right\}$$

Having an upper bound to the VC-dimension allows to obtain

Theorem 4.10 (Adaptation of Theorem 2.12 [64], see also [65])**.** *Let $(\mathcal{D},\mathcal{R})$ be a range space with $\mathrm{VC}(\mathcal{R}) \leq d$, let π be a distribution on $\mathcal{D}$ and $\mathcal{F}$ a family of functions defined*

on $\mathcal{D}$. Given, $\varepsilon, \delta \in (0,1)$, let

$$\ell = \frac{h}{\varepsilon^2}\left(d + \log\frac{1}{\delta}\right) \tag{4.4}$$

where h is an universal constant. Then, a bag $\mathcal{S} \subseteq \mathcal{D}$ of ℓ elements sampled independently according to π

$$SD(\mathcal{F}, \mathcal{S}) \leq \varepsilon$$

with probability at least $1 - \delta$.

The constant h is approximately 0.5 and it is *universal*, i.e., it does not depend on any parameter [66]. To understand the importance of Theorem 4.10 consider the following example. Assume that $\mathcal{R}$ contains a finite number of ranges and let π be the *uniform* distribution on $\mathcal{D}$, and $\mathcal{S}$ be a sample of points drawn uniformly and independently at random from $\mathcal{D}$. We can compute the sample size $|\mathcal{D}|$ needed to obtain a $\mathrm{SD}(\mathcal{R}, \mathcal{S})$ of at most ε with probability $1 - \delta$ by using the Hoeffding's [38] bound (see below) and the union bound [39]. Indeed, using the union bound we have that

$$\begin{aligned}&\mathbf{Pr}\left(\bigcup_{H\in\mathcal{R}} |\boldsymbol{a}_H(\mathcal{S}) - \mathbf{E}_{\mathcal{S}\sim\pi}\left[\boldsymbol{a}_H(\mathcal{S})\right]| > \varepsilon\right) \leq \sum_{H\in\mathcal{R}} \mathbf{Pr}\left(|\boldsymbol{a}_H(\mathcal{S}) - \mathbf{E}_{\mathcal{S}\sim\pi}\left[\boldsymbol{a}_H(\mathcal{S})\right]| > \varepsilon\right)\\ &\leq 2|\mathcal{R}|e^{-2|\mathcal{S}|\varepsilon^2}\end{aligned}$$

The above inequality tells us that to obtain a $\mathrm{SD}(\mathcal{R}, \mathcal{S})$ of at most ε with probability $1 - \delta$ we can safely stop sampling as soon as $2|\mathcal{R}|e^{-2|\mathcal{S}|\varepsilon^2} \leq \delta$, i.e., after the size of $\mathcal{S}$ is

$$|\mathcal{S}| = \frac{1}{2\varepsilon^2}\log\left(\frac{2|\mathcal{R}|}{\delta}\right) \tag{4.5}$$

where $\boldsymbol{a}_H(\mathcal{S}) = |H \cap \mathcal{S}|/|\mathcal{S}|$ and $\mathrm{SD}(\mathcal{R}, \mathcal{S}) = \sup_{H\in\mathcal{R}} |\boldsymbol{a}_H(\mathcal{S}) - \mathbf{E}_{\mathcal{S}\sim\pi}\left[\boldsymbol{a}_H(\mathcal{S})\right]|$.

Comparing this quantity with (4.4) clearly shows the multiple advantages of VC-dimension. For starters, the sample size suggested by (4.4) is smaller than the above as soon as the upper bound to the VC-dimension of the range space is smaller than $\log(2|\mathcal{R}|)$. Secondly, Theorem 4.10 holds for *any* distribution π and no assumption are made on it or any of its moments (e.g., on its variance). We use this property in Section 4.3.2 to develop efficient sampling-based randomized approximation algorithms for the temporal betweenness centrality.

4.3 MANTRA: temporal Betweenness Centrality Approximation through Sampling

4.3.1 Temporal Betweenness Estimator

In this section we present one *unbiased estimator*[4] for the $(\star)$-temporal betweenness centrality and we refer to Appendix A for an additional two estimators of the temporal betweenness that have been excluded from this chapter to maintain a cleaner exposition.

The ONBRA (**ob**) algorithm [4] uses an estimator defined over the sampling space $\mathcal{D}_{\mathbf{ob}} = \{(s, z) \in V \times V : s \neq z\}$ with uniform sampling distribution $\pi_{\mathbf{ob}}$ over $\mathcal{D}_{\mathbf{ob}}$, and family of functions $\mathcal{F}_{\mathbf{ob}}$ that contains one function $\widetilde{b}^{(\star)}_{\mathbf{ob}}(v) \to [0, 1]$ for each vertex v, defined as

$$\widetilde{b}^{(\star)}_{\mathbf{ob}}(v|s, z) = \frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}} \in [0, 1]$$

So far, this approach has been defined only for the shortest-temporal betweenness. In this chapter, we extend **ob** to shortest-foremost and prefix foremost temporal paths.

Lemma 4.11. *The function computed by **ob** is an unbiased estimator of the $(\star)$-temporal betweenness centrality.*

Proof.

$$\mathbf{E}\left[\widetilde{b}^{(\star)}_{\mathbf{ob}}(v|s, z)\right] = \sum_{\substack{s,z \in V \\ s \neq z \neq v}} \mathbf{Pr}\left((s, z)\right) \cdot \widetilde{b}^{(\star)}_{\mathbf{ob}}(v|s, z) = \sum_{\substack{s,z \in V \\ s \neq z}} \frac{1}{n(n-1)} \frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}}$$

□

We chose to display the results for **ob** throughout the chapter because among all the temporal betweenness estimators, it is the one that provides the best trade-off between *speed* and *quality* of approximation (see Appendix A for a detailed discussion).

4.3.2 Sample Complexity bounds

We present two bounds (Equation (4.6) and Theorem (4.13)) to the sufficient number of random samples to obtain an ε approximation of the $(\star)$-temporal betweenness centrality. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, with a straightforward application of Hoeffding's

[4] An estimator of a given parameter is said to be unbiased if its expected value is equal to the true value of the parameter.

inequality and union bound [39], it can be shown that $r = 1/(2\varepsilon^2)\log(2n/\delta)$ samples suffice to estimate the $(\star)$-temporal betweenness of every node up to an additive error ε with probability $1-\delta$. To improve this bound, we define the range space associated to the $(\star)$-temporal betweenness and its VC-dimension. Let $\mathcal{U} = \mathbb{TP}_{\mathcal{G}}^{(\star)}$, define the range space $\mathcal{R} = (\mathcal{D}, \mathcal{F}^+)$ where $\mathcal{D} = \mathcal{U} \times [0,1]$, and $\mathcal{F}^+$ is defined as follows: for a pair $(s,z) \in V \times V$ and a temporal path $tp_{sz} \in \mathcal{U}$ let $f_{(v,t)}(tp_{sz}) = f((v,t)|s,z) = \mathbb{1}[(v,t) \in \mathbf{Int}(tp_{sz})]$ be the function that assumes value 1 if the vertex appearance (v,t) is in the temporal path between s and z. Moreover, define the family of functions $\mathcal{F} = \{f_{(v,t)} : (v,t) \in V \times \mathcal{T}\}$ and notice that for each $f_{(v,t)} \in \mathcal{F}$ there is a range $R_{f_{(v,t)}} = \{tp_{sz} : tp_{sz} \in \mathcal{U}\}$. Next, define $\mathcal{F}^+ = \{R_{f_{(v,t)}} : f_{(v,t)} \in \mathcal{F}\}$. Now that we defined the range set for our problem, we can give an upper bound on its VC-dimension.

Lemma 4.12. *The VC-dimension of the range space $\mathcal{R}$ is $VC(\mathcal{R}) \leq \lfloor \log VD^{(\star)} - 2 \rfloor + 1$.*

Proof. Let $VC(\mathcal{R}) = d$, where $d \in \mathbb{N}$. Then, there is $S \subset \mathcal{D}$ such that $|S| = d$ and S is shattered by $\mathcal{F}^+$. For each temporal path $tp_{sz} \in \mathcal{U}$, there is at most one pair tp_{sz} in $\mathcal{S}$. By definition of shattering, each $tp_{sz} \in \mathcal{S}$ appears in 2^{d-1} different ranges in $\mathcal{F}^+$. Moreover, each tp_{sz} is in at most $|tp_{sz}| - 2$ ranges in $\mathcal{F}^+$, that is because $tp_{sz} \notin R_{f(v,t)}$ when $(v,t) \notin tp_{sz}$. Observe that $|tp_{sz}| - 2 \leq \mathrm{VD}^{(\star)} - 2$, gives $2^d \leq |tp_{sz}| - 2 \leq \mathrm{VD}^{(\star)} - 2$. Thus, $d - 1 \leq \log(\mathrm{VD}^{(\star)} - 2)$ since $d \in \mathbb{N}$, we have:

$$d \leq \lfloor \log \mathrm{VD}^{(\star)} - 2 \rfloor + 1 \leq \log(\mathrm{VD}^{(\star)} - 2) + 1$$

□

Given the VC-dimension of the range set $\mathcal{R}$ we apply Theorem 4.10 and we obtain that

$$r = \frac{h}{\varepsilon^2}\left[\lfloor \log \mathrm{VD}^{(\star)} - 2 \rfloor + 1 + \ln\left(\frac{1}{\delta}\right)\right] \tag{4.6}$$

samples suffice to compute an upper bound on the SD of at most ε with probability at least $1-\delta$ for the $(\star)$-temporal betweenness centrality. To improve this bound, we make use of Lemma 4.2 and notice that (as for the static case [46]) the $(\star)$-temporal betweenness centrality satisfies a form of negative correlation among the nodes. Moreover, the existence of a node v with high $(\star)$-temporal betweenness constraints the sum of the centrality measure over the remaining nodes to be at most $\rho^{(\star)} - \mathrm{b}_v^{(\star)}$. In other words, this suggests that the number of nodes with high $(\star)$-temporal betweenness cannot be arbitrarily large. Furthermore, as in [46], we assume that the maximum variance of the $(\star)$-temporal betweenness estimators $\widetilde{\mathrm{b}}_v^{(\star)}$ is bounded by some estimate $\hat{v}$ rather than the worst-case upper bound of $1/4$ considered in [45]. This implies that the estimates $\widetilde{\mathrm{b}}_v^{(\star)}$ are not bounded by the number of nodes in the temporal graph $\mathcal{G}$, but are tightly

constrained by the parameters $\rho^{(\star)}$ and $\hat{v}$. We are able to extend the results in [46] for the static scenario to the temporal setting for all the variants of temporal betweenness that can be computed in polynomial time and cover one of the problems left open by the authors. It follows that:

Theorem 4.13. *Let $\mathcal{F} = \{\widetilde{b}_v^{(\star)}, v \in V\}$ be a set of function from a domain $\mathcal{D}$ to $[0,1]$. Define $\hat{v} \in (0, 1/4]$ and $\rho^{(\star)} \geq 0$ such that*

$$\max_{v \in V} Var(\widetilde{b}_v^{(\star)}) \leq \hat{v} \quad \text{and} \quad \sum_{v \in V} b_v^{(\star)} \leq \rho^{(\star)}$$

Fix $\varepsilon, \delta \in (0,1)$, and let $\mathcal{S}$ be an i.i.d. sample taken from $\mathcal{D}$ of size

$$|S| \in \mathcal{O}\left(\frac{\hat{v}+\varepsilon}{\varepsilon^2} \ln\left(\frac{\rho^{(\star)}}{\delta \hat{v}}\right)\right)$$

It holds that $SD(\mathcal{F}, \mathcal{S}) \leq \varepsilon$ with probability $1-\delta$ over $\mathcal{S}$.

The proof of this theorem follows the one in [46] by considering the properties of the $(\star)$-temporal betweenness. For completeness, we show the adapted proof.

Proof. Given a sample S of size s, let E and E_v be the following events:

$$E = \text{“}\exists v \in V : |\mathrm{b}_v^{(\star)} - \widetilde{\mathrm{b}}_v^{(\star)}| > \varepsilon\text{”} \quad \text{and} \quad E_v = \text{“}|\mathrm{b}_v^{(\star)} - \widetilde{\mathrm{b}}_v^{(\star)}| > \varepsilon\text{”}$$

Applying the union bound we have that $\mathbf{Pr}(E) \leq \sum_{v \in V} \mathbf{Pr}(E_v)$. Define the functions $g(x) = x(1-x)$ and $h(x) = (1+x)\ln(1+x) - x$ for $x \geq 0$, and let $\hat{x}_1, \hat{x}_2, \hat{x}$ as

$$\hat{x}_1 = \inf\left\{x : \frac{1}{2} - \sqrt{\frac{\varepsilon}{2} - \frac{\varepsilon^2}{9}} \leq x \leq \frac{1}{2}, g(x)h\left(\frac{\varepsilon}{g(x)}\right)(2\varepsilon^2)\right\}$$

$$\hat{x}_2 = \frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}} \qquad \text{and,} \qquad \hat{x} = \min\{\hat{x}_1, \hat{x}_2\}$$

Moreover, using Hoeffding's and Bennet's bounds [61], Bathia and Davis bound on variance [67] and from the fact that $\max_{v \in V} \mathrm{Var}(f(v)) \leq \hat{v}$, it holds, for all $v \in V$

$$\mathbf{Pr}(E_v) \leq 2\min\left\{\exp\left(-2s\varepsilon^2\right), \exp\left(-s\gamma(\hat{v}, \mathrm{b}_v^{(\star)}, \varepsilon)\right)\right\} \doteq \eta(x)$$

where $\gamma(\hat{v}, \mathrm{b}_v^{(\star)}, \varepsilon) = \min\{\hat{v}, g(\mathrm{b}_v^{(\star)})\}h\left(\frac{\varepsilon}{\min\{\hat{v}, g(\mathrm{b}_v^{(\star)})\}}\right)$. Now we can write

$$\mathbf{Pr}(E) \leq \sum_{v \in V} E_v = \sum_{v \in V} \Phi(\mathrm{b}_v^{(\star)}) \tag{4.7}$$

Observe that the values of $\mathrm{b}_v^{(\star)}$ are not known a priori, thus it is not possible to compute the r.h.s. of Equation (4.7). However, we can obtain a sharp upper bound by using the constraints on the possible values of $\mathrm{b}_v^{(\star)}$ imposed by $\hat{v}$ and $\rho^{(\star)}$. As in [46] we define an appropriate optimization problem w.r.t. the unknown values of $\mathrm{b}_v^{(\star)}$. Let k_x be the number of nodes that we assume have $\mathrm{b}_v^{(\star)} = x$, define the optimization problem over the variables k_x as follows:

$$\begin{aligned} \max \quad & \sum_{\substack{x\in(0,1)\\ k_x>0}} k_x \Phi(x) \\ \text{subject to} \quad & \sum_{\substack{x\in(0,1)\\ k_x>0}} x k_x \leq \rho^{(\star)}, \\ & 0 \leq k_x \leq \tfrac{\rho^{(\star)}}{x}, k_x \in \mathbb{N} \end{aligned}$$

Observe that the first constraint follows from Lemma 4.2, and the second one directly by the definition of $\rho^{(\star)}$ itself. The values of the objective function of the optimal solution of the optimization problem give and upper bound to Equation 4.7. As for the static case [46], the optimization problem is a formulation of the Bounded Knapsack Problem [68] over the variables k_x in which items with label x are selected k_x times with unitary profit $\Phi(x)$ and weight x. Moreover, we consider the upper bound to the optimal solution given by the continuous relaxation, in which $k_x \in \mathbb{R}$, of the optimization problem (see Chapter 3 of [68]). Four our purpose, its enough to fully select the item with higher profit-weight ratio to fill the entire knapsack. Let $\overline{x} = \mathrm{argmax}_{x\in(0,1)}\{\Phi(x)/x\}$, the optimal solution of the continuous relaxation is $k_x = \rho^{(\star)}/\overline{x}, k_x = 0$, for all $x \neq \overline{x}$, while the optimal objective is $\frac{\rho^{(\star)}\Phi(\overline{x})}{\overline{x}} \geq \mathbf{Pr}\,(E)$. Moreover, observe that $\overline{x}$ always exists and $\Phi(x)/\overline{x} \in (0,1)$. The search of $\overline{x}$ can be simplified by exploiting the same approach used in [46] that leads to

$$\mathbf{Pr}\,(E) \leq \sup_{x\in(0,\min\{\hat{x}_1,\hat{x}_2\})} \left\{ \frac{\rho^{(\star)}\eta(x)}{x} \right\} \leq \sup_{x\in(0,\hat{x})} \left\{ \frac{\rho^{(\star)}\gamma(g(x),s,\varepsilon)}{x} \right\}$$

Setting $s \geq \sup_{(0,\hat{x})} \left\{ \ln(\frac{2\rho^{(\star)}}{x\delta})/(g(x)h(\frac{\varepsilon}{g(x)})) \right\}$ it holds that $\mathbf{Pr}\,(E) \leq \delta$. In order to approximate s, the r.h.s. of the equation can be computed using a numerical procedure [46, 55] obtaining the following approximation:

$$s \approx \frac{2\hat{v} + \frac{2}{3}\varepsilon}{\varepsilon^2}\left(\ln\left(\frac{2\rho^{(\star)}}{\hat{v}}\right) + \ln\left(\frac{1}{\delta}\right)\right) \in \mathcal{O}\left(\frac{\hat{v}+\varepsilon}{\varepsilon^2}\ln\left(\frac{\rho^{(\star)}}{\delta\hat{v}}\right)\right)$$

□

Since $\rho^{(\star)}$ correspond to the average number of internal nodes in $(\star)$-temporal paths in $\mathcal{G}$, it must be that $\rho^{(\star)} \leq D^{(\star)}$. In *all* the analyzed networks (see Figure 4.2 in Section 4.4)

this condition holds, thus this approach will need a smaller sample size compared to the VC-Dimension based one to obtain an ε-approximation of the $(\star)$-temporal betweenness.

We show how to estimate $\hat{v} = \sup_{f \in \mathcal{F}} \mathbf{Var}[f]$ using the empirical wimpy variance $\mathcal{W}_{\mathcal{F}}(\mathcal{S})$. Proposition 4.14 is an adaption of Proposition 4.3 [46] when we are using only a unique family of function $\mathcal{F}$ rather than a partition. For completeness we provide the adapted proposition and the proof.

Proposition 4.14 ([46])**.** *Let $\mathcal{F} = \{\widetilde{b}_v^{(\star)}, v \in V\}$ be a set of function from a domain $\mathcal{D}$ to $[0,1]$. And let $\mathcal{S} \subseteq \mathcal{D}$ be a sample of size r. Then, with probability at least $1-\delta$ it holds*

$$\sup_{f \in \mathcal{F}} \mathbf{Var}[f] \leq \mathcal{W}_{\mathcal{F}}(\mathcal{S}) + \frac{\ln(1/\delta)}{r} + \sqrt{\left(\frac{\ln(1/\delta)}{r}\right)^2 + \frac{\mathcal{W}_{\mathcal{F}}(\mathcal{S}) + \ln(1/\delta)}{r}} \tag{4.8}$$

Proof. By definition,

$$\sup_{f \in \mathcal{F}} \mathbf{Var}[f] = \sup_{f \in \mathcal{F}} \left\{ \mathbf{E}\left[f^2\right] - \mathbf{E}[f]^2 \right\} \leq \sup_{f \in \mathcal{F}} \mathbf{E}\left[f^2\right] \tag{4.9}$$

Thus we focus on bounding $\sup_{f \in \mathcal{F}} \mathbf{E}\left[f^2\right]$. By a straightforward application of Theorem 7.5.8 in [69] we have the claim of the lemma. □

4.3.3 Fast approximation of the characteristic quantities

According to Equation (4.6) and Theorem (4.13), the sample size needed to achieve a desired approximation depends on the *vertex diameter* and on the *average temporal path length* of the temporal graph. However, under the Strong Exponential Time Hypothesis (SETH), the $(\star)$-temporal diameter (thus the average $(\star)$-temporal path length) of a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ can not be computed in $\tilde{\mathcal{O}}(M^{2-\varepsilon})$[5] [70], which can be prohibitive for very large temporal graphs, so efficient approximation algorithms for these characteristic quantities are highly desirable. Some algorithms for the diameter approximation on temporal graphs have been proposed [70, 71]. However these techniques consider different temporal path optimality criteria [71], or have no theoretical guarantees [70]. In this chapter, we define a novel sampling-based approximation algorithm to efficiently obtain a high-quality approximation of $D^{(\star)}$ (thus, $VD^{(\star)}$) and $\rho^{(\star)}$ in $\tilde{O}(r \cdot M)$ where r is the number of samples used for the approximation. The high-level idea of the algorithm is that given a temporal graph $\mathcal{G}$, the

[5] With the notation $\tilde{\mathcal{O}}(\cdot)$ we ignore logarithmic factors.

sample size r, and the temporal path optimality $(\star)$, the algorithm performs r $(\star)$-temporal BFS visits [72] ($(\star)$-TBFS) from r random nodes and keeps track of the number of reachable pairs encountered at each hop along with the greatest hop performed. Once all the r visits have been completed, it computes the temporal diameter and other useful temporal measures using an approach based on the relation between the number of reachable pairs and the distance metrics (see Section 2.1.4). Formally Algorithm 2, given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$, a number of seeds r and the temporal path optimality of interest $(\star)$ draws a sample $\mathcal{S}$ of r random nodes from V, runs a temporal graph traversal from each $s_i \in \mathcal{S}$, and iteratively grows the ball $B(s_i, h) = \{v \in V : d(s_i, v) \leq h \wedge \mathbb{1}[s_i \rightsquigarrow v]\}$ for each h such that there exists at least one temporally reachable node that has not is $B(s_i, h-1)$ and is temporally reachable by at leas one node $v \in B(s_i, h-1)$. Once all the nodes in $\mathcal{S}$ have been processed, it computes an upper bound on the average number of internal nodes $\rho^{(\star)}$ (Equation 2.3), the temporal effective diameter $D_\tau^{(\star)}$ (Equation 2.2), the temporal diameter $D_{\text{LB}}^{(\star)}$ (Equation 2.1) and the temporal connectivity rate $\zeta^{(\star)}$ (Equation 2.4).

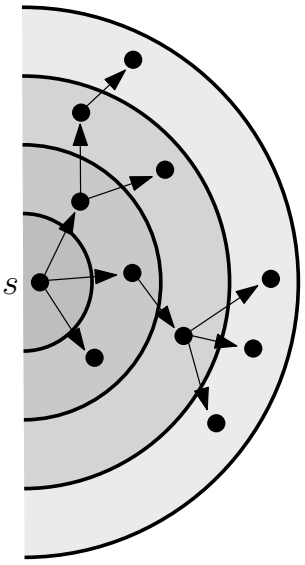


FIGURE 4.1: Example of a temporal BFS visit and of the ball centered in s.

Algorithm 2: $(\star)$-temporal (effective) diameter approximation

Data: Temporal graph $\mathcal{G}$, sample size r, temporal path optimality $(\star)$, and effective $(\star)$-temporal diameters's threshold τ

Result: Termpoal diameter lower bound $D_{\text{LB}}^{(\star)}$, effective diameter $D_\tau^{(\star)}$, temporal connectivity rate $\zeta^{(\star)}$, upper bound on $\rho^{(\star)}$

$R = [0, 0, \ldots, 0]$ // Nr. of temporally reachanble pairs at each hop

$D_{\text{LB}}^{(\star)} = 0; \rho^{(\star)} = 0; D_\tau^{(\star)} = -1$

for $k = 0$ *to* $r - 1$ **do**

  $u =$ sample u.a.r. a node from V

  $R, D_{\text{LB}}^{(\star)} = (\star)\text{-TemporalBFS}(\mathcal{G}, u, R, D_{\text{LB}}^{(\star)})$ // Update R and $D_{\text{LB}}^{(\star)}$

for $h = 0$ *to* $D_{LB} - 1$ **do**

  $R[h] = \frac{n}{r} \cdot R[h]$

  $\rho^{(\star)} = \rho^{(\star)} + (R[h] - R[h-1]) \cdot h$ // $R[-1]$ threaded as 0 when $h = 0$

$\rho^{(\star)} = \rho^{(\star)} / R[D_{\text{LB}}^{(\star)}]$

$D_\tau^{(\star)} = \min_h \left(h : \frac{R[h]}{R[D_{\text{LB}}^{(\star)}]} \geq \tau \right)$

$\zeta^{(\star)} = \frac{R[D_{\text{LB}}^{(\star)}]}{n(n-1)}$

return $D_{LB}^{(\star)}, D_\tau^{(\star)}, \zeta^{(\star)}, \rho^{(\star)}$

The approximation guarantees of our sampling algorithm strongly depend on "how temporally connected" a temporal graph is, i.e., depend on the temporal connectivity rate $\zeta^{(\star)}$. Intuitively, the higher the connectivity rate the higher the number of couples that are connected via at least one $(\star)$-temporal path and the better the approximation computed by Algorithm 2. Furthermore, the algorithm has the following theoretical guarantees:

Theorem 4.15. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and a sample of size $r = \Theta\left(\frac{\ln n}{\varepsilon^2}\right)$, the algorithm computes:*

I) $D^{(\star)}$ *with absolute error bounded by* $\frac{\varepsilon}{\zeta^{(\star)}}$*;*

II) $D_\tau^{(\star)}$ *with absolute error bounded by* $\frac{\varepsilon}{\zeta^{(\star)}}$*;*

III) $\rho^{(\star)}$ *with absolute error bounded by* $\frac{\varepsilon \cdot D^{(\star)}}{\zeta^{(\star)}}$*;*

IV) $\zeta^{(\star)}$ *with absolute error bounded by* ε*.*

with probability at least $1 - \frac{2}{n^2}$*.*

Proof. Let h be the $(\star)$-temporal effective diameter threshold, and

$$X_i^h = \frac{n \cdot \sum_{\{u : d(v_i,u) \leq h\}} \mathbb{1}[v_i \rightsquigarrow u]}{\sum_{\substack{u,v \in V \\ u \neq v}} \mathbb{1}[u \rightsquigarrow v]} = \frac{n \cdot |N(v_i, h)|}{|N(D^{(\star)})|}$$

observe that $X_i^h \in [0, \frac{1}{\zeta^{(\star)}}]$, and that X_i^h's expected value is

$$\mathbf{E}\left[X_i^h\right] = \sum_{v_i \in V} X_i^h \cdot \mathbf{Pr}\left(v_i\right) = \frac{\sum_{v_i \in V} |N(v_i, h)|}{|N(D^{(\star)})|} = \frac{|N(h)|}{|N(D^{(\star)})|}$$

Applying Hoeffding's inequality, with $\xi = \frac{\varepsilon}{\zeta^{(\star)}}$, we can approximate $\frac{|N(h)|}{|N(D^{(\star)})|}$ by $\frac{\sum_{i \in [r]} X_i^h}{r} = \frac{n \cdot \sum_{i \in [r]} |N(u, v_i)|}{r \cdot |N(D^{(\star)})|}$, and taking a sample of $r = \frac{\ln n}{\varepsilon^2}$ leads an error of $\frac{\varepsilon}{\zeta^{(\star)}}$ with probability of at least $1 - \frac{2}{n^2}$. Now we observe that the $(\star)$-temporal diameter can be defined in terms of the effective one by choosing $\tau = 1$, thus the bound holds also for the $(\star)$-temporal diameter. Next, define the random variable $X_i^D = \frac{|N(v_i, D^{(\star)})|}{n-1}$, and observe that $X_i^{D^{(\star)}} \in [0, 1]$. Observe that its expected value is exactly $\zeta^{(\star)}$:

$$\mathbf{E}\left[X_i^{D^{(\star)}}\right] = \sum_{v_i \in V} \frac{1}{n} \frac{|N(v_i, D^{(\star)})|}{n-1} = \frac{|N(D^{(\star)})|}{n(n-1)} = \zeta^{(\star)}$$

Again, applying Hoeffding's inequality, with $\xi = \varepsilon$, and taking $r = \frac{\ln n}{\varepsilon^2}$ sample nodes, we have an error bound of ε with high probability. Finally, define

$$X_i = \frac{n \cdot \sum_{u \in V} \mathbb{1}[v_i \rightsquigarrow u] \cdot d(v_i, u)}{\sum_{u,v \in V} \mathbb{1}[u \rightsquigarrow v]}$$

Observe that $X_i \in [0, \frac{D}{\zeta^{(\star)}}]$, and that its expectation is the average shortest temporal distance

$$\mathbf{E}[X_i] = \sum_{v_i \in V} X_i \cdot \mathbf{Pr}(X_i) = \frac{\sum_{v_i,u \in V} \mathbb{1}[v_i \rightsquigarrow u] \cdot d(v_i, u)}{\sum_{u,v \in V} \mathbb{1}[u \rightsquigarrow v]}$$

Finally, by using Hoeffding's inequality with $\xi = \frac{\varepsilon D}{\zeta^{(\star)}}$ and setting $r = \frac{\log n}{\varepsilon^2}$ we obtain that the error is of at most $\frac{\varepsilon D}{\zeta^{(\star)}}$ with probability $1 - \frac{2}{n^2}$. □

Observe that Algorithm 2 is general, it can be used to compute several notions of temporal diameter [37, 70–73].

4.3.4 The MANTRA Framework

In this section we introduce MANTRA[6], our algorithmic framework for the $(\star)$-temporal betweenness centrality estimation. MANTRA incorporates the bounds in Section 4.3.2 to compute an upper bound on the minimum sample size needed to approximate the SD of the $(\star)$-temporal betweenness and a state-of-the-art progressive sampling technique to speed-up the estimation process. The input parameters to MANTRA are: a temporal graph $\mathcal{G}$, a temporal path optimality $(\star) \in \{\text{sh}, \text{sfm}, \text{pfm}\}$[7], a target precision $\varepsilon \in (0, 1)$, a failure probability $\delta \in (0, 1)$, and a number of iterations for the bootstrap phase s'. The output is a vector $\mathcal{B}$ of pairs $(v, \widetilde{\text{b}}_v^{(\star)})$ for each $v \in V$, where $\widetilde{\text{b}}_v^{(\star)}$ is the estimate of $\text{b}_v^{(\star)}$ and $\mathcal{B}$ is probabilistically guaranteed to be an absolute ε approximation of the temporal betweenness. Formally:

Theorem 4.16. *Given a target accuracy $\varepsilon \in (0, 1)$ and a failure probability $\delta \in (0, 1)$, with probability at least $1 - \delta$ (over the runs of the algorithm), the output vector $\mathcal{B} = \{\widetilde{b}_v^{(\star)} : v \in V\}$ (obtained from a set of samples $\mathcal{S}$) produced by* MANTRA *is such that $SD(\mathcal{B}, \mathcal{S}) \leq \varepsilon$.*

Proof. Each coordinate in $\widetilde{\text{b}}_v^{(\star)}, v \in V$ is a sample mean (over a sample $\mathcal{S}$ of size r) of a specific function associated to an unbiased estimator for $\text{b}_v^{(\star)}$. Algorithm 3 stops if

[6] teMporAl betweeNness cenTrality appRoximation through sAmpling

[7] We point out that our approach is general, and can be extended to *every* definition of temporal betweenness centrality.

the number of the drawn samples is at least ω or if the supremum deviation bound ξ is at most ε. In other words it stops when the $SD(\mathcal{F}, \mathcal{S}) \leq \varepsilon$ and in both cases this is guaranteed (by Theorem 4.13 or Theorem 4.7) to happen with probability $1 - \delta$. □

Algorithm 3: MANTRA

Data: Temporal graph $\mathcal{G}$, $(\star)$ temporal path optimality, precision $\varepsilon \in (0, 1)$, failure probability $\delta \in (0, 1)$, bootstrap iterations s', and number of Monte Carlo trials c.

Result: Absolute ε-approximation of the $(\star)$-tbc w.p. of at least $1 - \delta$.

1 $\mathcal{B}, \mathcal{W} = [0, \ldots, 0] \in \mathbb{R}^n$ // tbc and wimpy variance arrays
2 $i, k = 0; \xi = 1; \mathcal{S}_0 = \{\emptyset\}$
3 $\omega, \hat{v} =$ DrawSufficientSampleSize$(\mathcal{G}, s', \delta/2)$
4 $\{s_i\}_{i \geq 1} =$ SamplingSchedule$(\omega, \hat{v}, \delta)$
5 $\boldsymbol{\lambda} = [[\cdot]]$ // Empty matrix
6 **while** *true* **do**
7 $\quad i = i + 1; k = (1.2 \cdot s_{i-1}) - s_{i-1}$
8 $\quad \mathcal{X} =$ DrawSamples$(\mathcal{G}, k)$// Draw k samples from the sample space $\mathcal{D}_{\mathbf{ob}}$
9 $\quad \mathcal{S}_i = \mathcal{S}_{i-1} \cup \mathcal{X}$
10 $\quad \boldsymbol{\lambda} =$ Add R.R.Vector$(k, \boldsymbol{\lambda})$ // Add a Rade. rnd. column of length c
11 $\quad \mathcal{B}, \mathcal{W}, \boldsymbol{\lambda} =$Update$(\star)$-TemporalBetweenness$(\mathcal{X}, \mathcal{B}, \mathcal{W}, \boldsymbol{\lambda})$
12 $\quad \tilde{R}, v_{\mathcal{F}} =$UpdateEstimates$(\mathcal{B}, \mathcal{W}, \boldsymbol{\lambda}, |\mathcal{S}_i|, k, c)$
13 $\quad R_k^c = \frac{1}{c} \sum_{l=1}^{c} \max_{v \in V} \left\{ \tilde{R}[v, l] \right\}$
14 $\quad \xi =$ ComputeSDBound$(R_k^c, v_{\mathcal{F}}, \delta/2^i, |\mathcal{S}_i|)$ // Compute Eq. 4.3 in Thm. 4.7
15 $\quad$ **if** $|\mathcal{S}_i| \geq \omega$ ***or*** $\xi \leq \varepsilon$ **then return** $\{(1/|\mathcal{S}_i|) \cdot \mathcal{B}[u] : u \in V\}$

Algorithm 3's execution is divided in two phases: the bootstrap phase (lines 3-4) and the estimation phase (lines 6-15). As a first step, MANTRA, computes an upper bound ω to the number of samples needed to achieve an ε approximation (line 2). The procedure runs s' independent $(\star)$-TBFS visits from s'[8] random couples of nodes (s, z) sampled from the population $\mathcal{D}_{\mathbf{ob}}$, estimates $\hat{v}$ and $\rho^{(\star)}$, and then plugs them in Theorem 4.13 to obtain ω. Subsequently, it infers the first element of the sample size $\{s_i\}_{i \geq 1}$ by performing a binary search between s' and ω to find the minimum s_1 such that Equation 4.3 (with R set to 0) is at most ε and terminates the bootstrap phase. Such approach gives an optimistic first guess of the number of samples to process for obtaining an ε-approximation [46]. Subsequently it continues with the estimation phase in which, at each iteration, it increases each s_i with a geometric progression [74], i.e., such that $s_i = 1.2 \cdot s_{i-1}$. Next, it proceeds by drawing uniformly at random $k = s_i - s_{i-1}$ couples of nodes (s, z) from $\mathcal{D}_{\mathbf{ob}}$ and subsequently updating the overall set of samples sampled so far (lines 7-9). Consequently, k new Rademacher random vectors are added as new columns to the matrix $\boldsymbol{\lambda}$ and k $(\star)$-TBFS visits are performed (line 11). Moreover, while iterating over the new sample $\mathcal{X}$ the temporal betweenness, wimpy variances and the

[8] In this chapter we use $s' = \log(1/\delta)/\varepsilon$.

values in $\boldsymbol{\lambda}$ are updated. After this step, the coefficients of Equation 4.3 and the new estimate on the SD, ξ, are computed (lines 12-13). As a last step of the while loop, the algorithm checks whether the desired accuracy ε has been achieved, i.e., whether the actual number of drawn samples is at least ω or ξ is at most ε (line 15). If at least one of the two conditions is met, MANTRA normalizes and outputs the current estimates $\mathcal{B}$. We conclude this section with the analysis of MANTRA's running time.

Theorem 4.17. *Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ and a sample of size r, MANTRA requires time $\tilde{\mathcal{O}}(r \cdot n \cdot T)$ and $\tilde{\mathcal{O}}(r \cdot M)$ to compute the shortest (foremost)-temporal and the prefix-foremost-temporal betweenness, respectively. Moreover,* MANTRA *requires $\mathcal{O}(n + M)$ space.*

Proof. MANTRA performs r truncated $(\star)$-TBFS visits and regularly check the stopping condition until convergence. The running times of the temporal traversals depend on the type of path optimality we consider. Moreover, each TBFS requires $\mathcal{O}(n \cdot T \cdot \log(n \cdot T))$ [3, 72] to compute the shortest (foremost) temporal betweenness and $\mathcal{O}(M \cdot \log M)$ to compute the prefix-foremost temporal betweenness. Furthermore, to compute and check the stopping condition of the progressive sampler we need roughly linear time in n. Thus MANTRA's running time is $\mathcal{O}(r \cdot n \cdot T \cdot \log(n \cdot T)) = \tilde{\mathcal{O}}(r \cdot n \cdot T)$ for the shortest and shortest foremost temporal betweenness and $\mathcal{O}(r \cdot M \cdot \log M) = \tilde{\mathcal{O}}(r \cdot M)$ for the prefix-foremost temporal betweenness. Finally, we observe that the space required by MANTRA is $\mathcal{O}(n + M + c \cdot n)$ observe that c is a fixed constant ($c = 25$), thus the overall needed space is $\mathcal{O}(n + M)$. □

Theorem 4.16 together with Theorem 4.17 provide theoretical evidence that MANTRA computes a rigorous estimation of the $(\star)$-temporal betweenness and that *scales* to the size of the input temporal graph. Moreover, it improves over the state-of-the-art approach ONBRA [4]. Indeed, given a sample of size r, ONBRA stores a $n \times r$ matrix to compute the absolute ξ-approximation[9] using the Empirical Bernstein Bound [34]. Thus, ONBRA may require an arbitrary large sample size (e.g. large matrix) to achieve a target absolute approximation ε, making the algorithm not ideal to analyze big temporal graphs.

4.4 Experimental Evaluation

In this section, we summarize the results of our experimental study on approximating the $(\star)$-temporal betweenness centrality in real-world temporal networks.

[9] ξ is the upper bound on the $SD(\mathcal{S}, \mathcal{F})$ obtained using the Empirical Bernstein Bound.

4.4.1 Experimental setting

We compare our novel framework with ONBRA [4]. For the sake of fairness, we adapted the original fixed sample size algorithm to use the same progressive sampling approach of our framework. Every time an element of the sampling schedule is consumed, the algorithm computes the upper bound ξ on the SD using the Empirical-Bernstein bound as in [4], if ξ is at most the given ε, it terminates, otherwise it keeps sampling. We set ONBRA's maximum number of samples to be equal to the VC-Dimension upper-bound in Section 4.3.2. We implemented all the algorithms in Julia exploiting parallel computing[10]. We chose to re-implement the exact algorithms [3] and ONBRA [4] because they have issues with the number of paths in the tested networks[11], causing overflow errors (indicated by negative centralities), and with the time labeling causing an underestimation of centralities [76]. Our implementation uses a sparse matrix representation of the $n \times |T|$ table used in [3, 4], making the implemented algorithms space-efficient and usable on big temporal graphs (for which the original version of the code gives out of memory errors). We executed all the experiments on a server running Ubuntu 16.04.5 LTS with one processor Intel Xeon Gold 6248R 32 cores CPU @ 3.0GHz and 1TB RAM. For every temporal graph, we ran all the algorithms with parameter $\varepsilon \in \{0.1, 0.07, 0.05, 0.01, 0.007, 0.005, 0.001\}$ chosen to have a comparable magnitude to the highest temporal betweenness values in the network (see $b^{(\star)}_{\max}$ in Table 4.1). This is a basic requirement when computing meaningful approximations[12]. Moreover, we use $\delta = 0.1$ and use $c = 25$ Monte Carlo trials as suggested in [46, 77]. Finally, each experiment has been ran 10 times and the results have been averaged.

4.4.2 Networks

We evaluate all the algorithms on real-world temporal graphs of different nature, whose properties are summarized in Table 4.1. The temporal networks come from three different domains:

Social networks. This domain includes most of the considered networks: College msg, Digg reply, Slashdot reply, Facebook Wall, Mathoverflow, SMS, Askubuntu, and Wiki Talk. These are social networks from different realms, where nodes correspond to users and temporal arcs indicate messages sent between them at specific points in time.

[10] Code available at: https://github.com/Antonio-Cruciani/MANTRA

[11] The overflow issue appears on *all* the transportation networks provided in [75].

[12] It is meaningless to compute an ε-approximation when the maximum centrality value is smaller than ε.

Contact networks. For the Topology network, nodes correspond to computers and temporal arcs indicate a contact between nodes at a specific time.

Transport networks. Bordeaux is part of the Kuala et al. [75] public transport networks collection. In such temporal graph, nodes are public transport stops and a temporal arcs indicate routes at a specific point in time. Because of their "inherent temporality", these networks are characterized by a big number of temporally connected nodes.

Figure 4.2 summarizes some key properties of the temporal graphs in Table 4.1. Furthermore, it also displays the approximation computed using Algorithm 2. Such approximations have been computed by running Algorithm 2 ten times using 256 random seeds and then averaging the results. In general, we observe that there is a big difference between $D^{(\star)}$ and $\rho^{(\star)}$, and that our sampling-based approximation algorithm provides very good estimates of these characteristic quantities even with a small sample size. Moreover, Figure 4.3 shows the comparison between All-Pairs-($\star$)-Temporal-Paths (i.e., the exact algorithm) and our approximation algorithm with a sample size of 256 random nodes. More precisely, Figure 4.3 displays the ratio between the running time (in seconds) of the exact algorithm and Algorithm 2. We observe that the approximation algorithm provides a huge speed-up without compromising the quality of the approximation of these fundamental quantities.

Data set	n	$\lvert\mathcal{E}\rvert$	$\lvert T\rvert$	ζ	$b_{\max}^{(\text{pfm})}$	$b_{\max}^{(\text{sh})}$	$b_{\max}^{(\text{sfm})}$	**Type**	**Source**
College msg	1899	59798	58911	0.5	0.0718	0.0319	0.0365	D	[44]
Digg reply	30360	86203	82641	0.02	0.0019	0.0015	0.0016	D	[48]
Slashdot	51083	139789	89862	0.07	0.0128	0.0074	0.0085	D	[48]
Facebook Wall	35817	198028	194904	0.04	0.0034	0.0024	0.0028	D	[48]
Topology	16564	198038	32823	0.53	0.0921	0.0654	0.0681	U	[49]
Bordeaux$^{\bullet}$	3435	236075	60582	0.84	0.1210	0.1383	0.1269	D	[75]
Mathoverflow	24759	390414	389952	0.33	0.0522	0.0282	0.0287	D	[44]
SMS	44090	544607	467838	0.008	0.0019	0.0010	0.0012	D	[44]
Askubuntu	157222	726639	724715	0.169	0.0214	0.0156	0.0154	D	[44]
Super user	192409	1108716	1105102	0.21	0.0261	0.0165	0.0182	D	[44]
Wiki Talk	1094018	6092445	5799206	0.069	0.0089	0.0155	0.0153	D	[49]

TABLE 4.1: The data sets used in our evaluation, where ζ indicates the exact temporal connectivity rate, $b_{\max}^{(\star)}$ the maximum ($\star$)-temporal betweenness centrality (type D stands for directed and U for undirected). $\bullet$ indicates that we need to use BigInt data type instead of Unsigned Int128 to count the number of shortest (foremost)-temporal paths to avoid overflows.

4.4.3 Experimental Results

Efficiency and Scalability. In our first experiment, we compare the average execution times, sample sizes and allocated memory of MANTRA and ONBRA. Here we show the results on the data sets in Table 4.1 for the prefix-foremost and shortest temporal

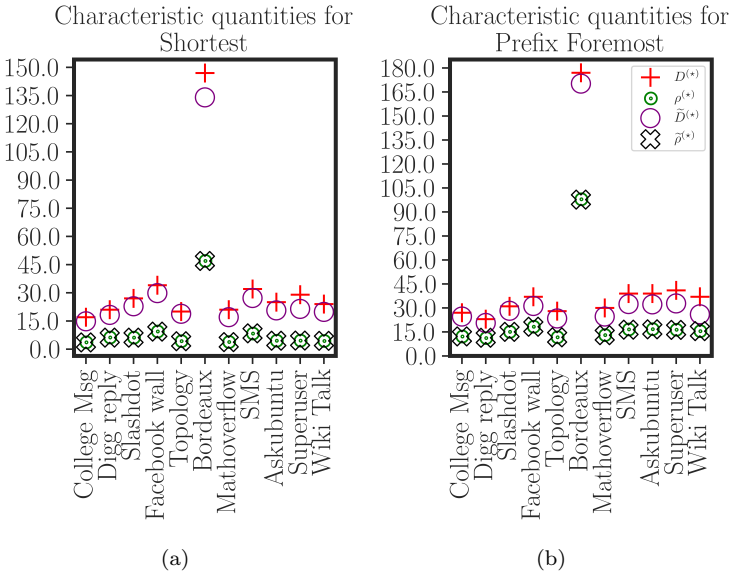


FIGURE 4.2: Comparison between the temporal diameter and the average number of internal nodes for the Shortest (foremost) and Prefix-Foremost temporal path optimalities. The approximation has been computed (over 10 runs) using our sampling algorithm using 256 random seed nodes.

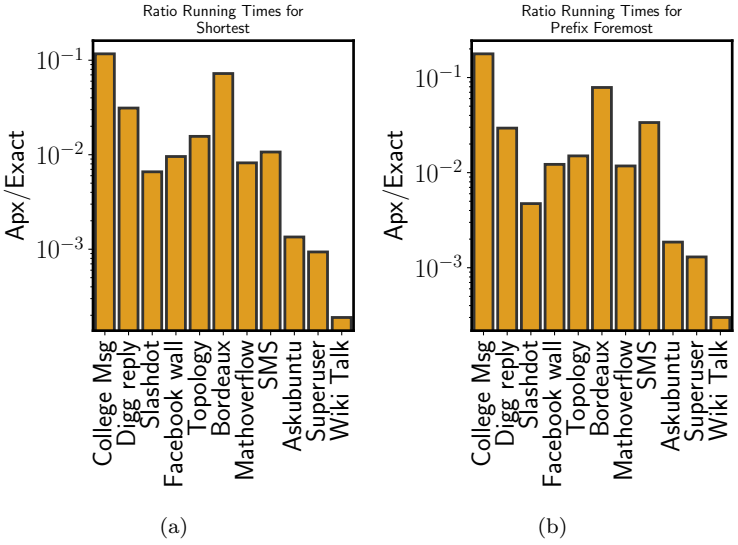


FIGURE 4.3: Ratio between the running time of the Exact algorithm for the temporal distance-based metrics and our approximation algorithm for the Shortest (foremost) and Prefix-Foremost temporal path optimalities. The approximation has been computed (over 10 runs) using our sampling algorithm using 256 random seed nodes.

betweenness, for a subset of $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$ and we refer to Appendix A for the complete battery of experiments. We chose to display the results for the pfm temporal path optimality because it is the one for which the analyzed graphs have the highest characteristic quantities (see Figure 4.2). Thus, under this setting, the tested algorithms will need a bigger sample size and potentially a higher amount of memory.

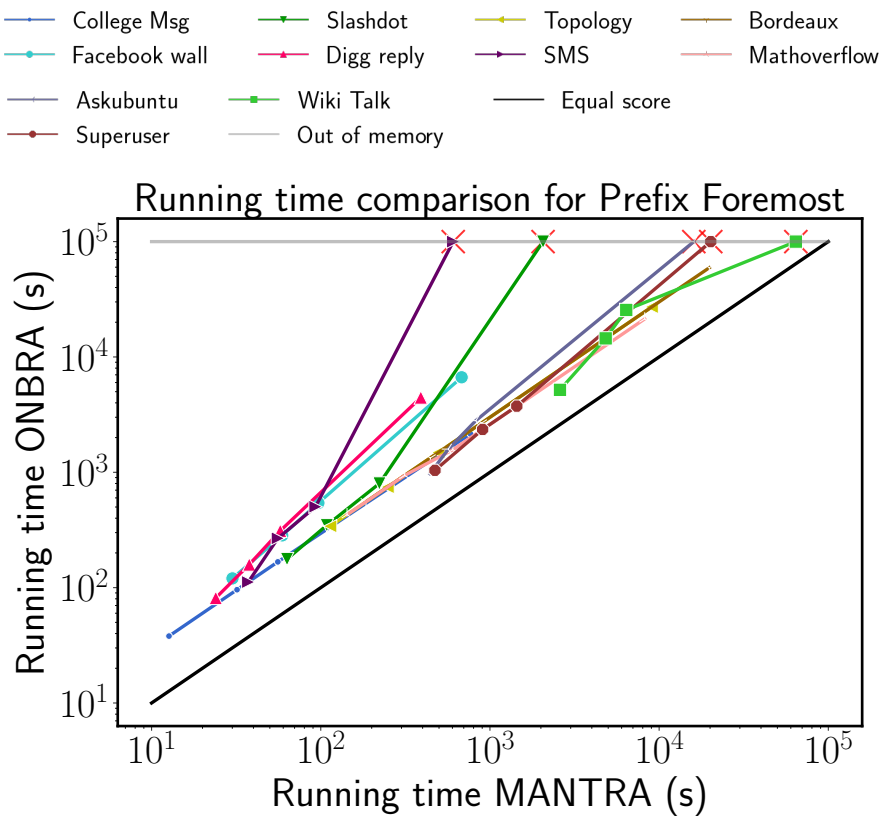
This somehow provides an intuitive "upper bound" on the algorithms performances in terms of efficiency and scalability. Furthermore, we also show the experiments for the sh temporal betweenness. That is one of the most computationally intensive temporal path optimalities for the temporal betweenness among the one considered in this chapter.

Moreover, the experiments for the sfm temporal betweenness follow similar trends of the ones displayed int his chapter. Figure 4.4 shows the comparison of the running times (in seconds) for the pfm and sh temporal betweenness. We observe that MANTRA leads the scoreboard against its competitor on *all* the tested networks. Our novel framework is at least three times faster than ONBRA. Such speedup is mainly due to the smaller sample size needed to terminate. Furthermore, Figure 4.5 shows that MANTRA requires a smaller sample size (at least three times smaller) to converge. This early convergence, in practice, does not affect the approximation quality and leads to very good temporal betweenness approximations (see the next experiment). Furthermore, the number of samples needed by MANTRA varies among temporal graphs, with a strong dependence on $b_{\max}^{(\star)}$. A potential cause of the difference in the sample sizes between the two algorithms may depend on the use of the Empirical Bernstein bound. Such bound (as the VC-Dimension one) is agnostic to any property of the analyzed temporal network, thus results in a overly conservative guarantees. This suggests that *variance-adaptive* bounds are preferable to compute data-dependent approximations [46], and that exploiting correlations among the nodes through the use of the c-MCERA leads to refined guarantees. Moreover, we point out that ONBRA does not scale well as the target absolute error ε decreases. Indeed, the memory needed by ONBRA increases drastically as the target absolute error decreases (see Figure 4.6) to the point of giving out of memory error for big temporal networks such as Slashdot, SMS, Askubuntu, Superuser, and Wiki Talk. This can lead to major issues while computing meaningful ε-approximations, especially under the setting in which the maximum temporal betweenness $b_{\max}^{(\star)}$ is very small (for which we need to choose an ε value of at most $b_{\max}^{(\star)}$[13]). Unfortunately, this is not an uncommon feature of real-world temporal networks. Indeed, as shown by ζ and $b_{\max}^{(\star)}$ in Table 4.1 they tend to be very sparse. This experiment, suggests that MANTRA is preferable for analyzing big temporal networks up to an arbitrary small absolute error ε.

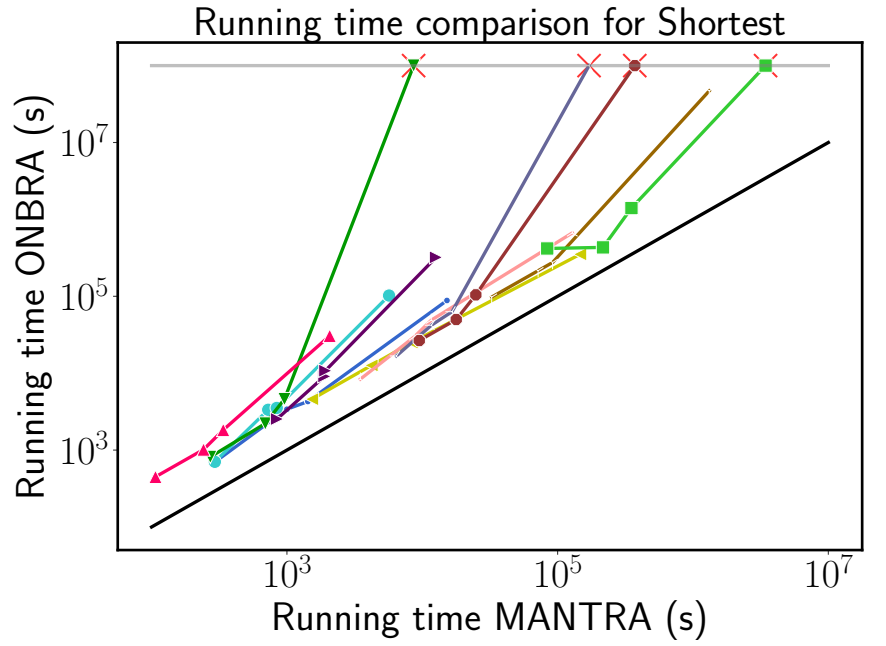
Comparison with the exact algorithms scores and running times. As a first step in our second experiment, we investigate the accuracy of the approximations provided by MANTRA by computing the exact temporal betweenness centrality of all the nodes of the temporal network in Table 4.1 and measuring the SD over all the ten runs. Figure 4.7 supports our theoretical results, as we always get a SD of at most the given ε.

[13]We recall that $b_{\max}^{(\star)}$ can be efficiently approximated in the bootstrap phase of our framework.

Moreover, we point out that the exact algorithms for the shortest (foremost) temporal betweenness required a time that spanned from several hours (e.g. for SMS) to days (for Askubuntu, and Superuser $\approx$ a week) and weeks (for Wiki Talk $\approx$ a month). Instead, MANTRA completes the approximation in reasonable time. Figure 4.8(a) shows the relation between the sample size and the running time of our framework. While, Figure 4.8(b) shows the amount of time needed by MANTRA to provide the absolute ε-approximation in terms of percentage of exact algorithm's running time. We display the running times on the biggest temporal graphs for the sh temporal betweenness because is one of the "critical" temporal path optimalities that requires longer times to be computed (see Theorem 4.17). We can conclude that our framework is well suited to quickly compute *effective* approximations of the temporal betweenness on very large temporal networks.

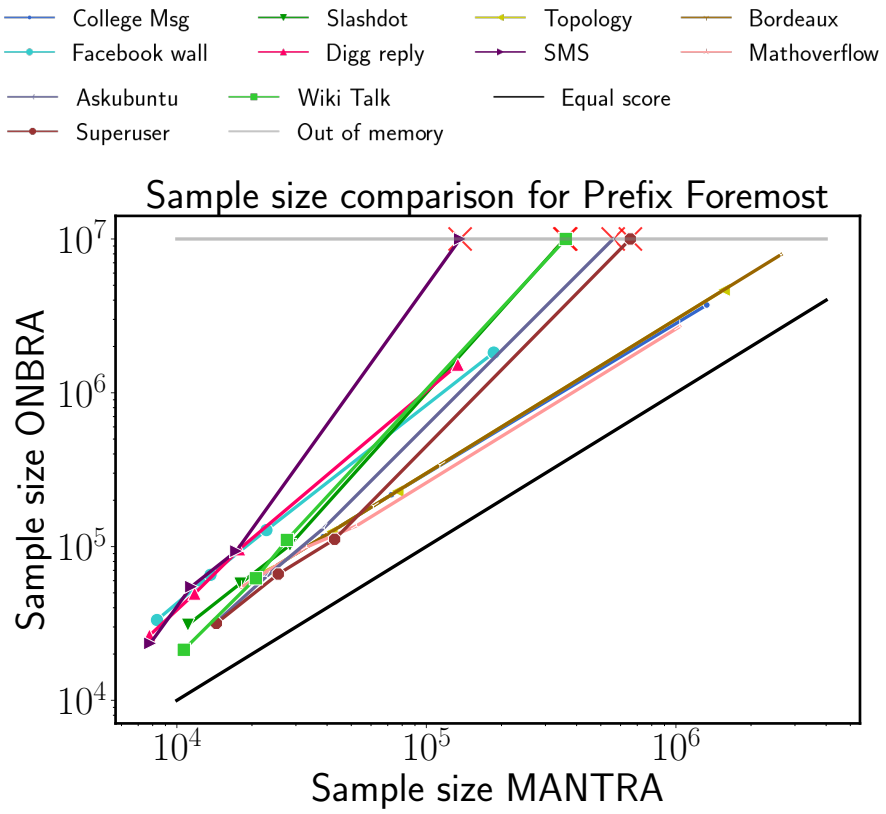


(a)

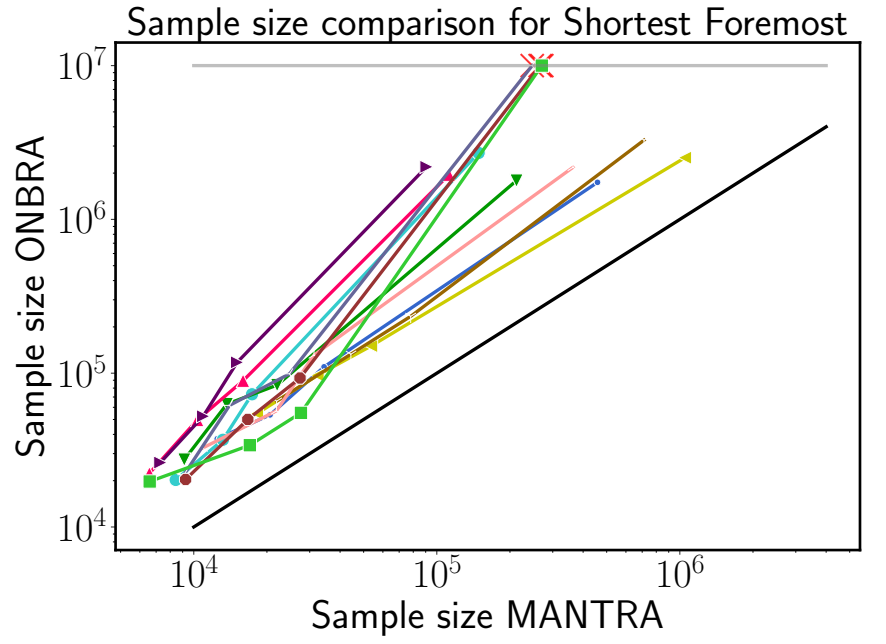


(b)

FIGURE 4.4: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$. Comparison between the running times for the pfm **(a)**, and sh **(b)** of ONBRA and MANTRA. The black line indicates that the two algorithms require the same amount of time, gray line (followed by a red mark) indicates that the algorithm required more than 1TB of memory to run on that data set with that specific ε value.

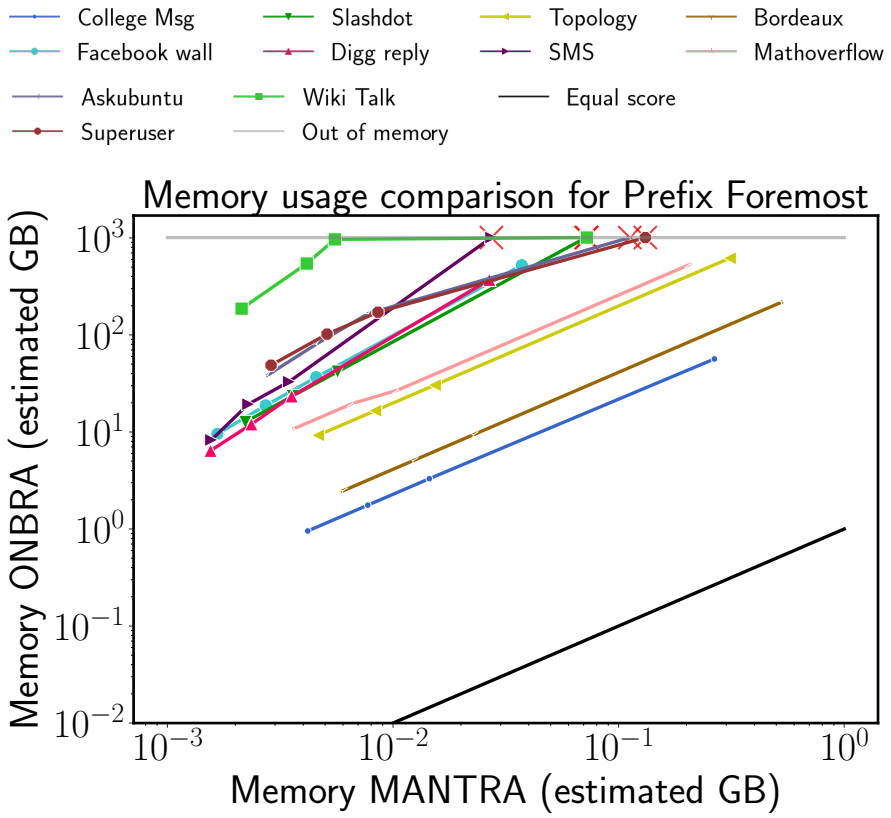


(a)

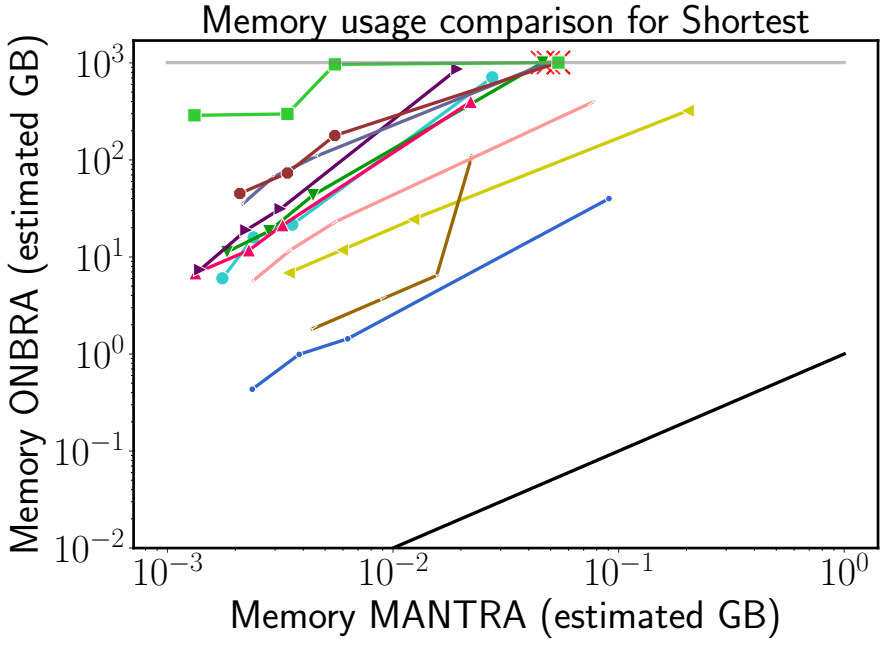


(b)

FIGURE 4.5: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$. Comparison between the sample sizes for the pfm **(a)** and sh **(b)**, of ONBRA and MANTRA. The black line indicates that the two algorithms require the same amount of samples, gray line (followed by a red mark) indicates that the algorithm required more than 1TB of memory to run on that data set with that specific ε value.

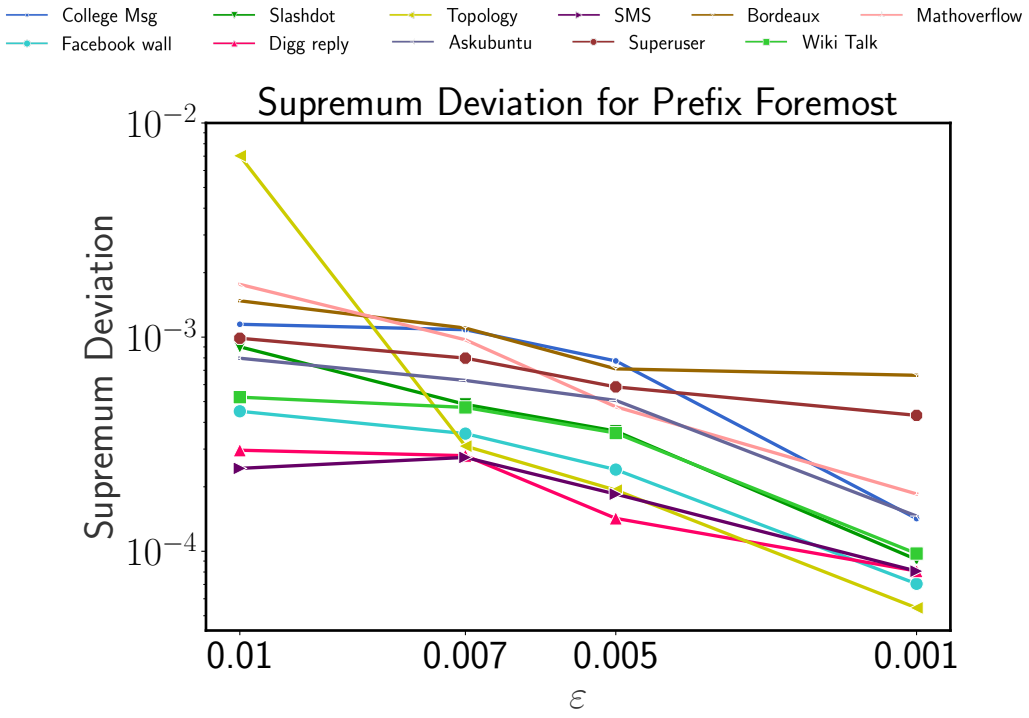


(a)

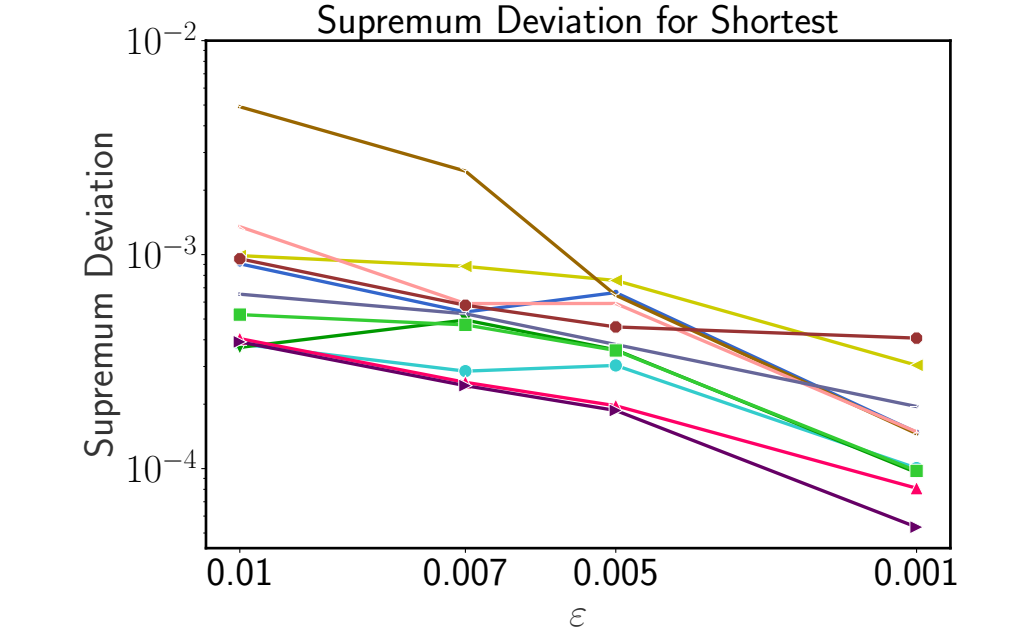


(b)

FIGURE 4.6: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$. Comparison between the allocated memory for pfm **(a)**,and sh **(b)**, of ONBRA and MANTRA. The black line indicates that the two algorithms require the same amount of memory, gray line (followed by a red mark) indicates that the algorithm required more than 1TB of memory to run on that data set with that specific ε value.

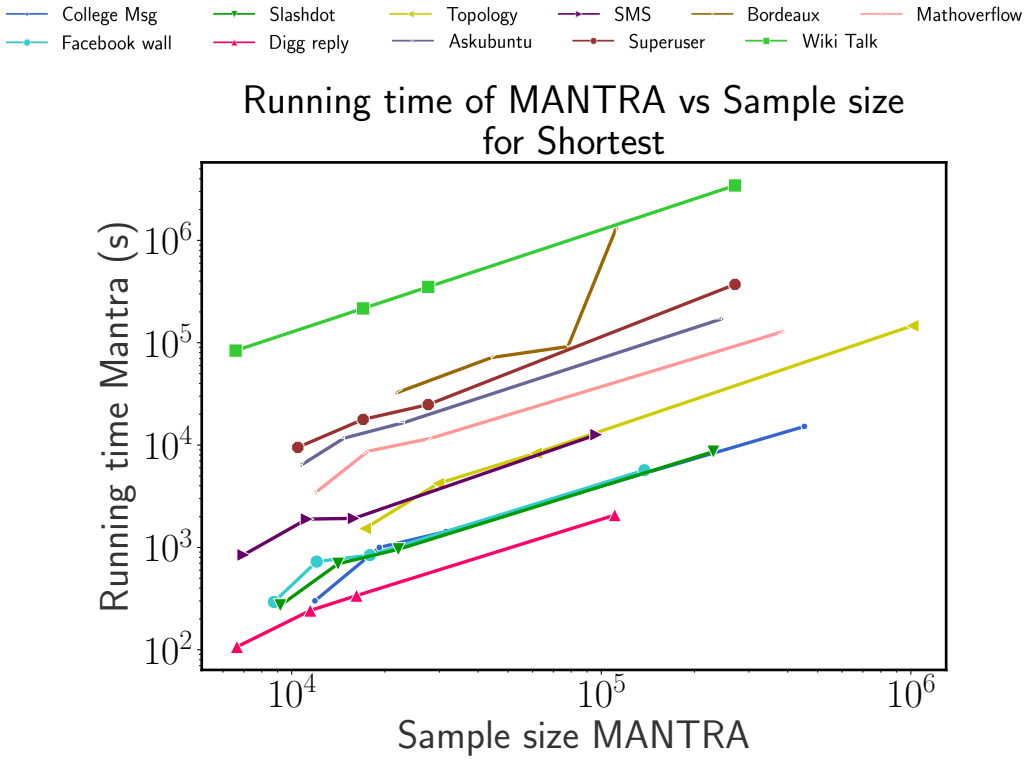


(a)

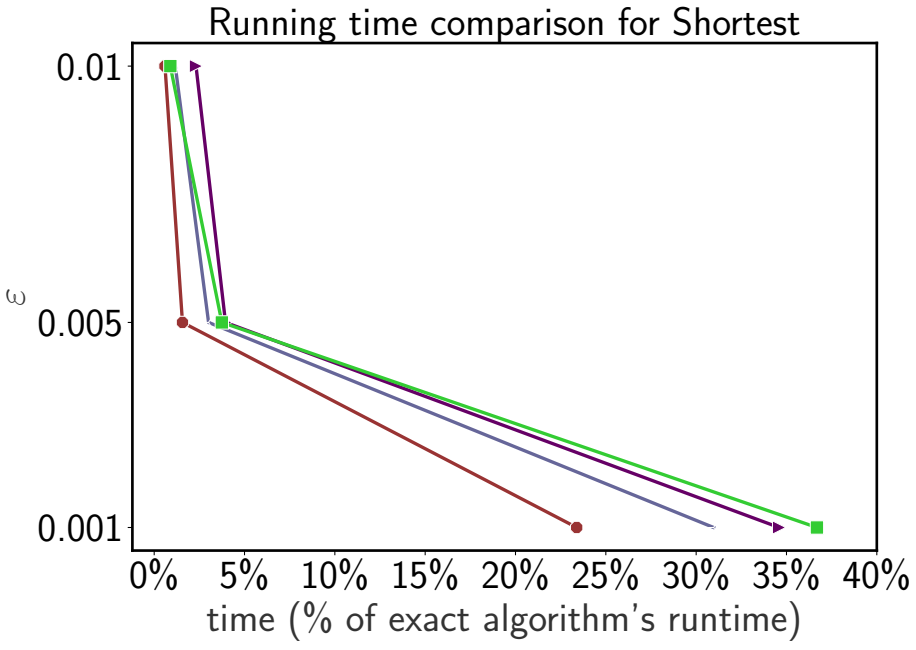


(b)

FIGURE 4.7: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$. Comparison between the supremum deviation for pfm **(a)**,and sh **(b)**, of MANTRA.



(a)



(b)

FIGURE 4.8: **(a)** Relation between the running time and the sample size of MANTRA for the shortest temporal betweenness with ε as in Figure 4.4. **(b)** Comparison between MANTRA and the exact algorithm running times for the shortest temporal betweenness on the biggest temporal networks.

Chapter 5

Expansion and Flooding time on Dynamic Random Regular Expanders

In this chapter we propose several dynamic random graphs that are inspired by the network formation process in the Bitcoin protocol. We run extensive simulations to measure the "flooding time" in the dynamic graphs, i.e., how long it takes a message starting at a random node to reach all, or almost all, the nodes.

5.1 Introduction

Bitcoin is a cryptocurrency proposed in 2008 by an unknown person or group of people under the pseudonym of Satoshi Nakamoto [78]. The system is built using a clever combination of a few classical cryptographic concepts: cryptographic hash functions [79], digital signature schemes [80], and hash-cash style proof-of-work [81]. Nodes participating in the Bitcoin system are connected toward an unstructured peer-to-peer network [82] running on top of the Internet. The first version of the software was released by Satoshi Nakamoto in January 2009. The most widely used implementation coming from that initial release, Bitcoin-core [83], is currently under active development. In this chapter we are concerned with dynamic graph models inspired by the network formation process of the Bitcoin P2P network. We refer the reader interested in a complete description of the Bitcoin system to [84, 85].

After an initial bootstrap in which they rely on DNS seeds for node discovery, nodes running the Bitcoin-core implementation turn to a fully-decentralized policy to regenerate

their neighbors when their degree drops below the configured threshold [86]. Each node has a "target out-degree value" and a "maximum degree value" (respectively 8 and 125, in the default configuration) and it locally stores a large list of (ip addresses of) "active" nodes. Every time the number of current neighbors of a node is below the configured target value it tries to create new connections with nodes sampled from its list. The list stored by a node is initially started with nodes received in response to queries to DNS seeds, then it is periodically advertised to its neighbors and updated with the lists advertised by the neighbors. Hence, in the long run each node samples its out-neighbors from a list formed by a "sufficiently random" subset of all the nodes of the network.

While most of the nodes of the network can be easily discovered [87], the existence of an edge between two nodes is only known by the two endpoints. The topology of the Bitcoin network is thus hidden by the network formation protocol. Indeed, discovering the network structure has been recently an active research topic [88, 89].

5.1.1 Our contribution.

RAES (Request a link, then Accept if Enough Space) [90] is a directed random graph model defined by three parameters $n \in \mathbb{N}, d \in \{1, \ldots, n-1\}, c > 1$, in which each one of n nodes has out-degree exactly d and in-degree at most cd. The random graph is generated according to the following discrete random process: The graph starts with no edges, and at every round each node u with out-degree $d_u^{\text{out}} < d$ picks $d - d_u^{\text{out}}$ nodes *uniformly at random (u.a.r.)* (with repetitions) and, for each such node v, u "requests" a directed link (u, v); If a node v receives a number of link-requests that would make its in-degree larger than cd, then v rejects all requests received in the current round, otherwise v accepts all requests of the round. The process terminates when all nodes have out-degree d (and in-degree at most cd).

The RAES model can be seen as a simplified version of the network-formation process implemented in Bitcoin-core [86]. However, it lacks one of the crucial aspects of the real network: the *dynamics*, i.e., the fact that nodes can join and leave the network at any time and edges can be faulty. In this chapter we consider an undirected version of RAES and we extend the random graph model in two ways, both of them generating *dynamic* random graphs that perpetually evolve. We run extensive simulations of both models to grasp the "stationary" structural properties of the dynamic random graphs and to measure the time it takes a message generated from one node to reach all (or almost all) the nodes.

In the first model, *edge-dynamic RAES (E-RAES)*, we add an "edge-evolution" parameter $p \in [0, 1]$ with the following role: At every round, each accepted edge disappears

with probability p. A detailed description of this model is presented in Section 5.2.1. Since the set of nodes of the graph is fixed while the set of edges evolve in discrete rounds, the dynamic random graph is a sequence $\{G_t = (V, E_t) : t \in \mathbb{N}\}$ where the distribution of the edges at round t only depends on the set of edges at round $t-1$. In order to empirically measure when the dynamic random graph can be considered stable, we compute the sequence of spectral gaps γ_t of the transition matrices P_t of the snapshots G_t of the dynamic graph and we consider that the dynamic graph is in a stable regime when γ_t remains in a sufficiently small interval for a sufficiently large window of consecutive rounds. The spectral gap of the transition matrix is also a measure of how "well-connected" a graph is and the results of the simulations show that the model generates, on average, sequences of graphs that are well-connected even with large values for the edge disappearing rate p. Indeed, even when a large fraction of edges disappear at any round, one single step of the RAES procedure is typically sufficient to rebuild a well-connected graph.

In the second model, *vertex-dynamic RAES (V-RAES)*, we add two "node-evolution" parameters, $\lambda \in \mathbb{R}^+$ and $q \in [0, 1]$, with the following roles: At every round t, $N_\lambda(t)$ new nodes enter the network, where $N_\lambda(t)$ is a Poisson random variable with rate λ, and each node leaves the network with probability q, independently of the other nodes. As soon as a new node joins the network, it starts requesting links to the nodes already in the network; one round later, i.e., when the presence of the new node has been revealed to the network, the node also starts receiving incoming link requests from other nodes; when a node leaves the network, all its incident links disappear. A detailed description of this model is given in Section 5.2.2. In the V-RAES model the network evolution is a sequence of random graphs $\{G_t = (V_t, E_t) : t \in \mathbb{N}\}$ in which both the set of nodes and the set of edges are random sets at any round. It is easy to see that the expected number of nodes in the graph converges to λ/q, if we consider it *before* the node-leaving step, and to $\lambda(1-q)/q$ if we consider it *after* the node-leaving step.

The dissemination protocol in Bitcoin-core is a gossip-based flooding: When a node receives a valid transaction, it announces it to all its neighbors (see, e.g., https://en.bitcoin.it/wiki/Network#Standard_relaying). As far as we know there are recent proposals to modify the dissemination mechanism aiming at improving network bandwidth usage [91] or limiting de-anonymization attacks [92], but to the best of our knowledge they have not been implemented in Bitcoin-core so far (see, e.g., https://github.com/bitcoin/bips/blob/master/bip-0330.mediawiki). In both our models, E-RAES and V-RAES, we simulate the flooding process and we measure the *flooding time*, i.e., how long it takes a message starting at a random node to reach all (or almost all) the nodes of the graph.

For the E-RAES model, the results of the simulations show that the flooding time is short (i.e., compatible with a logarithmic growth, as a function of the number of nodes), for every value of the edge-disappearance rate p. For the V-RAES model, the results of the simulations show that, as long as the fraction of nodes that leave the network at any round is not too large, e.g., if it stays below 70%, a message starting at a random node typically quickly reaches nearly all of the nodes.

We also simulate a combination of the two models, in which nodes join and leave the network as in the V-RAES and edges can be faulty as in the E-RAES. In this chapter we present the set of results obtained by simulating the models with only a few representative ranges for parameters d and c that determine the neighborhood size of the nodes: the smallest possible values for which the underlying graph turns out well-connected and the default values used in the main Bitcoin implementation. However, we remark that simulations with different values of d and c exhibit similar qualitative behavior. We implemented all the models in Python[1].

Notice that the topology of the evolving random graphs generated according to our models is almost surely quite far from the evolving topology of the real Bitcoin network, since each node of the real network can autonomously decide how many neighbors it wants to have and how to try to connect to them, and typically nodes choose different strategies based on their different needs. However, the topology of the evolving random graphs generated according to our models is probably close to the topology that the Bitcoin network would have if all full-nodes used the Bitcoin-core implementation with the default parameters. The study of our models thus allows us to give evidence of the long-term stability of the network generation process implemented in Bitcoin-core. This, in turn, gives an indication about the long-term stability of a large part of the real network without revealing its topology.

As noted in [93], most design decisions implemented at the network layer of permissionless blockchains imply some tradeoffs that typically are not yet well-understood. In this respect, the results of our simulations suggest that the default values used in the main Bitcoin implementation that determine the size of the neighborhood of a full-node could be safely reduced by most of the full-nodes to save network bandwidth without compromising the stability of the network.

5.1.2 Related work

The topology of the Bitcoin network is hidden by the network formation protocol. However several approaches in the last decade proved effective in revealing some portion of

[1]Software available at: https://github.com/Antonio-Cruciani/dynamic-random-graph-generator.

the network. Miller et al. [94] developed a set of tools and an infrastructure to discover the public Bitcoin network. Their approach was subsequently made ineffective by an update in the Bitcoin protocol. Neudecker et al. [89] proposed a timing analysis that is able to infer the network topology with a sufficient degree of precision. Delgado-Segura et al. [88] proposed a new approach to reconstruct the network structure and tested it on the Bitcoin *testnet* network revealing a network with 733 nodes and 6090 edges, with an average degree of 16.6 and with most of the nodes having between 7 and 14 neighbors. As far as we know it has never been tested on the Bitcoin *main* network.

Peer-to-Peer (P2P) networks received a lot of attention in the last twenty years and several (static and dynamic) network models have been proposed so far. A random network model for unstructured P2P networks was introduced and analyzed by Panduragan et al. [95]. Their model was inspired by the Gnutella P2P network and is based on the existence of a *host server* that maintains a *cache* of constant size with addresses of nodes accepting connections that can be reached at any time by other nodes. In [96] the authors introduced a class of dynamic graphs called *Dynamic Networks with Churn* (in short, *DNC*) where both node insertion/deletion and edge evolution are considered. The authors assume that the dynamic graph consists of a sequence of *d-regular expander graphs*; for the purpose of that paper, such an assumption is justified by the results in [97], where the authors presented a distributed protocol that guarantees the maintenance of a bounded degree topology that, with high probability, contains an expander subgraph whose set of vertices has size $n - o(n)$, where n is the "stable" network size. In [98] the authors defined two churn processes: in the first one, at every round a new node is added to the network while no node leaves it; in the second one, the size of the vertex set is n and when a new node joins the network the oldest node leaves it. The authors designed a protocol where each node u starts $c \cdot m$ independent random walks (containing the ID-label of the node) until they are picked up by new nodes joining the network, that connects to the peers that contributed to the tokens. The resultant dynamic topology is shown to keep diameter $\mathcal{O}(\log n)$ and to be fault-tolerant against adversarial deletion of both edges and vertices. The tokens in the graphs must be circulating at each time step in order to ensure that they are well-mixed; this implies that the rate at which new nodes can join the system is limited, as they must wait while the existing tokens mix before they can use them. Bagchi et al. [99] studied the number of adversarial and random faults that an expander graph can tolerate while preserving approximately the same expansion factor and a linear number of nodes. Becchetti et al. [90] introduced and analyzed the *RAES* network formation model, in which after a logarithmic number of rounds the network evolution terminates in a state in which every node has a specified out-degree and in-degree upper bounded by a constant. In a recent work Becchetti et al. [100] introduced and studied a similar model in which nodes can

also join and leave the network, but the in-degree of the nodes is not upper bounded by a constant.

Several well known problems have been studied in the context of dynamic networks: (byzantine) agreement, search and storage, (byzantine) leader election, expander maintenance, information spreading, membership management (we refer the reader to [101] for a survey). For information spreading, early works considered gossip-based broadcast algorithms (see, e.g., [102]). However, for privacy oriented P2P networks, such as the Bitcoin P2P network, some of these algorithms have been shown to expose the network to privacy vulnerabilities [103] and motivated the design of more sophisticated information spreading algorithms with low overhead as well as strong resistance to de-anonymization attacks [91, 92, 104]. Being able to randomly select other peers as new neighbors to maintain a random-graph like overall structure (low diameter, bounded degree, etc.) is another critical issue in such networks that has been extensively studied (see, e.g., [105–107]).

5.2 The models and the problem

A *dynamic graph* $\mathcal{G}$ is a sequence of graphs $\mathcal{G} = \{G_t = (V_t, E_t) : t \in \mathbb{N}\}$ where the sets of nodes and edges can change at any discrete round. If they change randomly, we call the corresponding random process a *dynamic random graph*. In this section we introduce two dynamic random graph models, that we call *Edge-dynamic RAES (E-RAES)* and *Vertex-dynamic RAES (V-RAES)*, that extend the RAES model introduced in [90].

5.2.1 Edge-dynamic RAES (E-RAES)

The E-RAES model is defined by four parameters, n, d, c, and p, where $n \in \mathbb{N}$ is the number of nodes, $d \in \mathbb{N}$ is the *minimum target degree*, $c \cdot d$ with $c \geqslant 1$ is the *maximum acceptable degree*, and $p \in [0, 1]$ is the *edge-failure probability*. The set of n nodes is fixed, while the set of edges evolves, at each round, in three steps. In the first step, each node with less than d neighbors connects with randomly chosen nodes in order to reach its minimum target degree; in the second step, each node with more than $c \cdot d$ neighbors, disconnects from randomly chosen neighbors in order to remain within its maximum acceptable degree; in the third step, each edge disappears with probability p, independently of the other edges.

Starting from an arbitrary initial graph $G_0 = (V, E_0)$.
At each round $t \in \mathbb{N}$:

Step 1: For each node $u \in V$, let N_u^1 be the set of neighbors of u at the beginning of Step 1. If $|N_u^1| < d$ then u samples $d - |N_u^1|$ nodes from the set $V \setminus N_u^1$, independently and u.a.r. with replacement, and connects to them.

Step 2: For each node $u \in V$, let N_u^2 be the set of neighbors of u at the beginning of Step 2. If $|N_u^2| > c \cdot d$ then u samples $|N_u^2| - (c \cdot d)$ neighbors from the set N_u^2, independently and u.a.r. with replacement, and disconnects from them.

Step 3: Each edge $\{u, v\}$ currently in the graph disappears with probability p, independently of the other edges.

The E-RAES model defines a Markov chain with the set of all graphs with n nodes as state space. It is not difficult to see that the chain is aperiodic and that the empty graph is a *recurrent* state (see, e.g., Chapter 1.5 in [108] for some background). Hence, if we consider the recurrent class containing the empty graph, the Markov chain defined by the E-RAES model starting at the empty graph will converge to a stationary distribution π. From a theoretical point of view, it would be interesting to analyze the expansion properties of the stationary random graph (i.e., a random graph sampled according to the stationary distribution) and to estimate the *mixing time* of the Markov chain, i.e., the time it takes the distribution of the chain starting at the empty graph to get close to the stationary distribution. However, a theoretical analysis of the mixing time appears quite challenging due to the complexity of the Markov chain: for example, next paragraph we observe that the chain is not *reversible*, thus it not possible to apply the large body of tools developed for the analysis of reversible chains (see, e.g., [109]). In Section 5.3 we propose an empirical convergence criterion, we present the results on the expansion properties of the snapshots of the dynamic graph obtained by simulating the E-RAES, and the results on the time it takes a message starting at a random node to reach all the nodes.

E-RAES non-reversibility. A Markov chain $\{X_t\}_t$ with state space Ω and transition matrix P is *reversible* if a probability distribution π over Ω exists such that for every pair of states $x, y \in \Omega$ the following *detailed balanced equation* holds: $\pi(x)P(x, y) = \pi(y)P(y, x)$. The analysis of reversible Markov chains can take advantage of several mathematical tools (see, e.g., [109]) that typically are not available for non-reversible chains. In this appendix we observe that the Markov chain defined by the E-RAES model is non-reversible.

The E-RAES model defines a Markov chain $\mathcal{M}$ where the state space Ω is formed by all the graphs with n nodes and, for two states/graphs $x, y \in \Omega$, $P(x, y)$ is the probability to reach state y from state x following the three steps of the E-RAES model, as defined in Section 5.2.1. Observe that, given two arbitrary states $x, y \in \Omega$, in general it is not possible to reach state y starting from state x with a sequence of states $x = z_0, z_1, \ldots, z_k = y$ such that $P(z_i, z_{i+1}) > 0$ for every $i = 0, 1, \ldots, k-1$. However, if we restrict the state space to the subset $\hat{\Omega} \subseteq \Omega$ of all the states that can be reached starting from the empty graph $G_0 = (V, \emptyset)$, it is easy to see that the Markov chain restricted to state space $\hat{\Omega}$ is *irreducible* and *aperiodic* (see, e.g., Chapters 1.5-1.7 in [110] for some background). Hence, there is a unique stationary distribution $\hat{\pi}$ over $\hat{\Omega}$ such that, starting from any state $x \in \hat{\Omega}$, $P^t(x, \cdot)$ converges to $\hat{\pi}$ as t goes to infinity (see, e.g., Theorem 4.9 in [110]).

If, by contradiction, there was a probability distribution π over $\hat{\Omega}$ satisfying the detailed balanced equation $\pi(x)P(x, y) = \pi(y)P(y, x)$, then π would be stationary for P (see, e.g., Proposition 1.20 in [110]) and, for the uniqueness of the stationary distribution, we would have $\pi = \hat{\pi}$. Notice that, since $\hat{\pi}$ is the stationary distribution of an irreducible Markov chain with state space $\hat{\Omega}$, then $\hat{\pi}(x) > 0$ for every $x \in \hat{\Omega}$. However, it is not difficult to find two states $x, y \in \hat{\Omega}$ such that $P(x, y) > 0$ and $P(y, x) = 0$. Indeed, consider two graphs x and y such that in both of them each node has degree between d and cd, and the set of edges in y is a subset of the set of edges in x. Clearly $P(x, y) > 0$, since $P(x, y)$ is at least as large as the probability that exactly all the edges in x that are not in y disappear during Step 3 of the E-RAES, and $P(y, x) = 0$, since in y every node has degree between d and cd thus no new edges are created during Steps 1 and 2 of the E-RAES model. Hence for such two states it must be $0 < \hat{\pi}(x)P(x, y) \neq \hat{\pi}(y)P(y, x) = 0$.

5.2.2 Vertex-dynamic RAES (V-RAES)

The V-RAES model is defined by four parameters, λ, d, c, and q, where $\lambda > 0$ is the *arrival rate* of new nodes, d and $c \cdot d$ are the *minimum target degree* and the *maximum acceptable degree* as described in the E-RAES model, and $q \in [0, 1]$ is the *node-leaving probability*. At each round t the graph evolves in four steps. In step *zero* $N_\lambda(t)$ new nodes join the graph, where $N_\lambda(t)$ is a Poisson random variable with rate λ. In step *one*, each node with less than d neighbors (hence, including the $N_\lambda(t)$ newly arrived ones) connects with randomly chosen nodes among those that are in the graph at the current round and were also present in the graph at the previous round (hence, excluding the $N_\lambda(t)$ newly-arrived nodes). In step *two*, each node with more than $c \cdot d$ neighbors, disconnects from randomly chosen neighbors in order to remain within its maximum acceptable degree.

In step *three*, each node u disappears with probability q, independently of the other nodes (all edges incident to u disappear as well).

Starting from an arbitrary initial graph $G_0 = (V_0, E_0)$.
At each round $t \in \mathbb{N}$:

Step 0: $N_\lambda(t)$ new nodes join the graph, where $N_\lambda(t)$ is a Poisson random variable with rate λ.

Step 1: For each node u, let N_u^1 be the set of neighbors of u at the beginning of Step 1. If $|N_u^1| < d$ then u samples $d - |N_u^1|$ nodes from the set $(V_t \setminus N_\lambda(t)) \setminus N_u^1$, independently and u.a.r. with replacement, and connects to them.

Step 2: For each node u, let N_u^2 be the set of neighbors of u at the beginning of Step 2. If $|N_u^2| > c \cdot d$ then u samples $|N_u^2| - (c \cdot d)$ neighbors from the set N_u^2, independently and u.a.r. with replacement, and disconnects from them.

Step 3: Each node u disappears with probability q, independently of the other nodes, together with its incident edges.

The size of the vertex set V_t in the V-RAES model converges to $\lambda(1-q)/q$, if measured at the end of the round, and it converges to λ/q if measured at the end of step *two* of the round, i.e., before the node-leaving step. Indeed, consider the following informal argument: Let us name f_t the expected number of nodes at round t, then $f_t = (f_{t-1} + \lambda)(1-q)$, if computed at the end of the round, since in expectation λ new nodes join the network at round t and each node in the graph remains in the network with probability $(1-q)$. Solving the recurrence with initial condition $f_0 = 0$ gives $f_t = \lambda \sum_{i=1}^{t} (1-q)^i = \lambda \left(1 - q - (1-q)^{t+1}\right)/q$, that converges to $\lambda(1-q)/q$ for t that goes to infinity. More formally, the size of the vertex set is actually a Markovian queue $M \backslash G \backslash \infty$ and it converges to a Poisson random variable of rate λ/q (see, e.g., [95]). In Section 5.4 we present the results of the simulations of the V-RAES model.

5.2.3 Preliminaries

Spectral gap. Let $G = (V, E)$ be an undirected graph with no self-loops. The *transition* matrix of a simple random walk on G (we will refer to it as the transition matrix of G) is the $|V| \times |V|$ matrix $P = D^{-1}A$, where A is the adjacency matrix of G and D is the diagonal matrix whose entries are the degrees of the nodes (for each node $u \in V$, $D(u,u)$ is the degree of u in G). It is well-known that P is *reversible*, all its eigenvalues

are real and they belong to the interval $[-1, 1]$ and the largest eigenvalue is $\lambda_1 = 1$. Moreover, the second largest eigenvalue $\lambda_2 < 1$ if and only if G is connected. In this case the *spectral gap* $\gamma = 1 - \lambda_2$ is a measure of how quickly the random walk converges to its stationary distribution (the largest the spectral gap the fastest the convergence rate). In the following paragraph we recall that the spectral gap is also a measure of how "well-connected" the underlying graph G is.

Expanders and spectral gaps. A graph $G = (V, E)$ with $|V| = n$ nodes is a $(1+\delta)$-*vertex expander*, for some $\delta > 0$, if for every set S of size at most $n/2$, the neighborhood $N(S) = \{v \in V : \{u, v\} \in E \text{ for some } u \in S\}$ has size at least $(1 + \delta)|S|$. It is known that, for a regular graph G, if we define $\gamma = 1 - \max\{\lambda_2, |\lambda_n|\}$, where λ_2 and λ_n are respectively the second-largest eigenvalue and the smallest eigenvalue of the transition matrix P, then the graph G is a $(1 + \gamma)$-vertex expander (see e.g. Chapter 4 in [111]). Thus, larger values of the spectral gap γ of the transition matrix P correspond to better expansion of the underlying graph G.

Let $\mathcal{G} = \{G_t = (V_t, E_t) : t \in \mathbb{N}\}$ be a dynamic (random) graph. For the purpose of this paper, we measure how well-connected are the snapshots G_t of the dynamic graph by computing the spectral gaps of their transition matrices.

Flooding. To measure the time it takes a message sent by a node to reach all (or a large fraction of) nodes we use the following *flooding process*. Let $\mathcal{G} = \{G_t = (V_t, E_t) : t \in \mathbb{N}\}$ be a dynamic random graph. The flooding process over $\mathcal{G}$ starting at round t_0 from the initiator $u_0 \in V_{t_0}$ is the sequence of (random) sets of nodes $\{I_t : t \in \mathbb{N}\}$ such that: $I_t = \emptyset$ for $t < t_0$; $I_{t_0} = \{u_0\}$; and for $t > t_0$

$$I_t = (I_{t-1} \cup N(I_{t-1})) \cap V_t$$

where $N(I_{t-1})$ is the set of nodes in $V_{t-1} \setminus I_{t-1}$ that in graph G_{t-1} have at least one neighbor in I_{t-1}

$$N(I_{t-1}) = \{v \in V_{t-1} \setminus I_{t-1} : \{u, v\} \in E_{t-1} \text{ for some } u \in I_{t-1}\}$$

We say that I_t is the subset of *informed* nodes at round t. If at some round t all nodes currently in the network are informed, i.e. $I_t = V_t$, we say that the flooding is *complete*. The *flooding time* is the number of rounds between t_0 and the first round t such that $I_t = V_t$.

5.3 E-RAES simulations

In this section, we present the results of the simulations of the E-RAES model. On the one hand we are interested in the structural properties of the snapshots of the evolving graphs, on the other hand we want to measure how long it takes a message starting at one node to reach all the others. To evaluate the structural properties, we use the *spectral gap* of the transition matrix; to evaluate the speed of information spreading we use the *flooding time* (see Section 5.2.3).

In Section 5.3.1 we define an empirical converge criterion that we will use to decide the starting round for the simulations computing the average spectral gap of the snapshots of the evolving graph and the average duration of the flooding process. In Section 5.3.2 we present the results of the simulations for the spectral gap and in Section 5.3.3 those for the flooding process.

We present only the results for some representative parameters d and c: the minimum target degree $d = 4$ is small enough to guarantee that the resulting snapshots of the evolving graph are quite "sparse"; the value $c = 1.5$ makes the nodes quite "inflexible" about their target degree (each node only accepts to have degree 4, 5, or 6). Despite these strict requirements about the graph structure, our simulations show that the random process quickly stabilizes on a stationary regime, where the snapshots of the graph are often very good expanders, even for large values of the edge-failure probability p. We remark that results qualitatively very similar to those presented for $d = 4$ and $c = 1.5$ appear for different values of d and c.

5.3.1 Convergence criterion

We want to study how fast the information spreads from a node to all the other nodes when the network evolution is *stationary*. In order to decide the starting round for the flooding process, we need a criterion to establish when the network evolution reaches stationarity. In principle, it would be possible to give theoretical bounds on the number of rounds needed to reach stationarity by analyzing the *mixing time* of the Markov chain induced by the E-RAES model; however, as we mentioned in Section 5.2, the analysis of such a Markov chain appears far from easy. For the purpose of this paper, we use a heuristic criterion based on the stabilization of the spectral gap. We set an $\varepsilon > 0$ and we declare that the graph stabilizes when the spectral gap remains in a range of width 2ε for $\log n$ consecutive rounds. More formally, at the generic round $t \geqslant \log n$, if all spectral gaps $\gamma_{t-\log n}, \gamma_{(t-\log n)+1}, \ldots, \gamma_t$ differ from γ_t for at most ε then we declare that the dynamic random graph mixed. The choice of ε is dynamically computed by the

following rule: we simulate a long-run of the evolving graph for 100 rounds and we set ε as the mean absolute deviation [112] of the non-zero values.

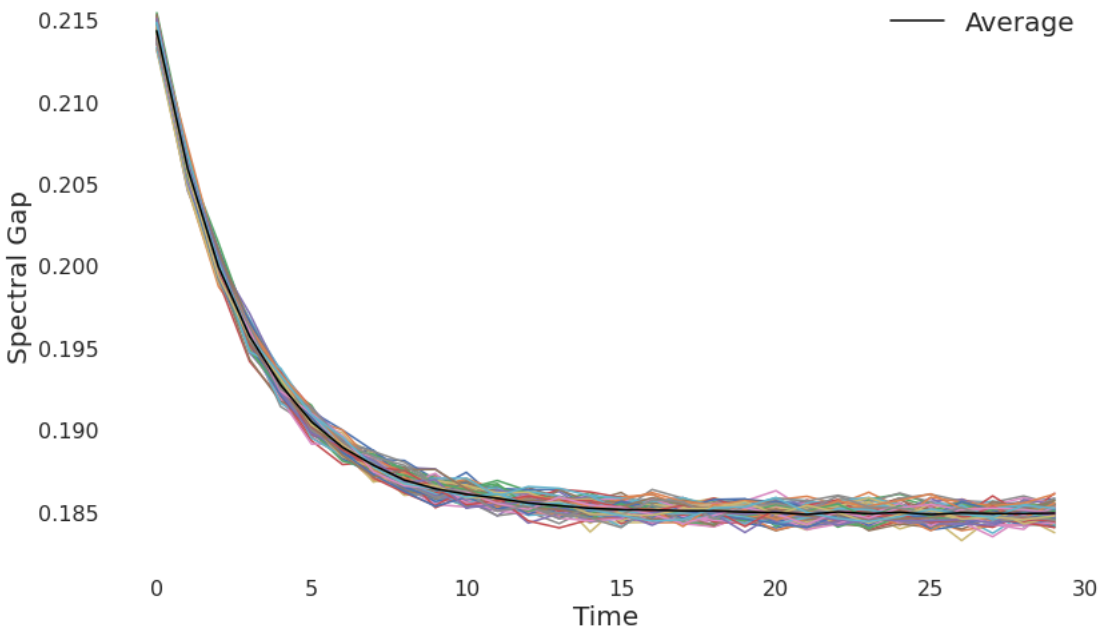


FIGURE 5.1: The initial evolution of the spectral gap and its stabilization for the dynamic graph with 2^{15} nodes, $d = 4$, $c = 1.5$, and starting from the empty graph. The spectral gap at each round is computed before the edge-failure step. Each line in the picture plots one out of one hundred executions. The bold black line plots the average of the spectral gaps computed at each round, over all the executions.

Figure 5.1 shows a representative sample of the evolution of the spectral gap during the first rounds of the E-RAES model with $n = 2^{15}$ nodes, $d = 4$, $c = 1.5$, and edge-failure probability $p = 0.1$, starting from the empty graph. The spectral gap of the snapshots of the evolving graph is computed before the edge failure step. The picture shows that, after about 15 rounds, the spectral gap stabilizes with very small oscillations, from round to round in each execution and with little difference from one execution to another.

5.3.2 Average spectral gap in the long run

To measure the expansion properties of the typical snapshot of the evolving graph, we simulate the E-RAES model and compute the average of the spectral gaps of the snapshots. The table and the plot in Figure 5.2 show the results of the simulations for different values for the number of nodes n and edge-failure probability p. Each number in the table is the average over 100 runs of 100 rounds each. We computed the spectral gap both before and after the edge-failure step.

Average spectral gap								
Nodes \ p		**0.0**	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1.0**
1024	**B**	0.345	0.19	0.208	0.234	0.269	0.317	0.345
	A	0.345	0.157	0.0	0.0	0.0	0.0	0.0
2048	**B**	0.342	0.189	0.206	0.232	0.268	0.315	0.342
	A	0.342	0.155	0.0	0.0	0.0	0.0	0.0
4096	**B**	0.341	0.187	0.204	0.231	0.267	0.314	0.341
	A	0.341	0.144	0.0	0.0	0.0	0.0	0.0
8192	**B**	0.341	0.186	0.204	0.23	0.266	0.313	0.341
	A	0.341	0.09	0.0	0.0	0.0	0.0	0.0
16384	**B**	0.34	0.186	0.203	0.23	0.265	0.313	0.34
	A	0.34	0.053	0.0	0.0	0.0	0.0	0.0
32768	**B**	0.34	0.185	0.203	0.23	0.265	0.313	0.34
	A	0.34	0.006	0.0	0.0	0.0	0.0	0.0

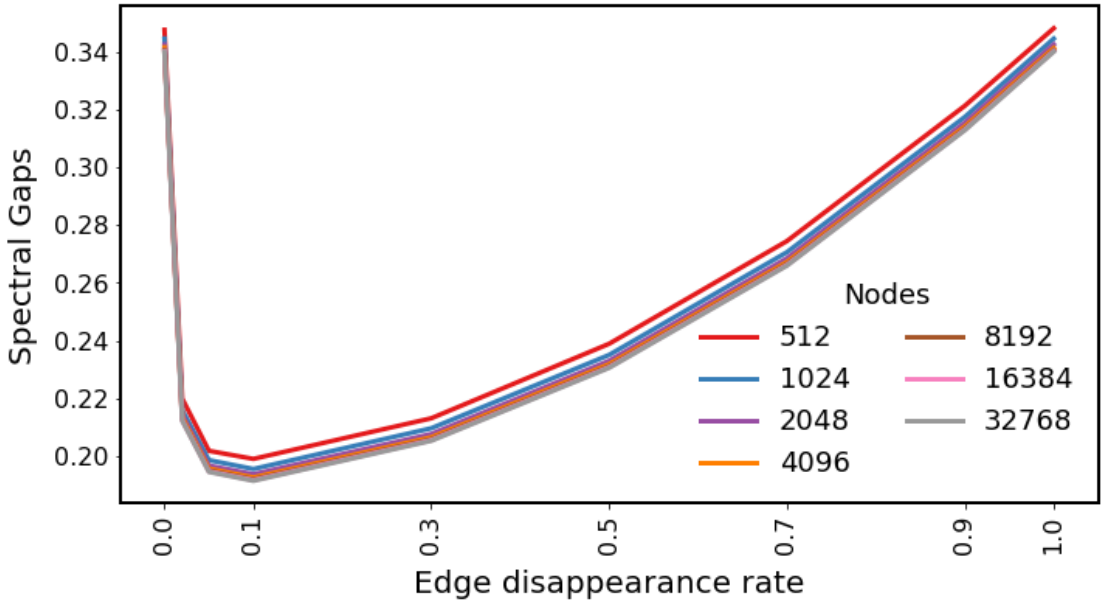


FIGURE 5.2: Average spectral gap for E-RAES of 100 runs of 100 rounds each, for $d = 4$, $c = 1.5$, and increasing values for number of nodes n and edge-failure probability p. The spectral gap is computed before (B) and after (A) the edge-failure step.

For small values of p, e.g., $p = 0.1$, the first column of the table in Figure 5.2 shows that the snapshots are on average connected (the spectral gap is non-zero) even *after* the edge failure step. Although the differences between the spectral gaps computed before and after the edge-failure step, that increases with the number of nodes, indicates that after the edge-failure step the resulting graph tend to become a much weaker expander, even when only 10% of the edges disappear on average.

For larger values of p the snapshots of the graph after the edge faults turn out to be mostly disconnected (spectral gap equals to zero), however the spectral gap computed

before the edge-failure step indicates that every time the graph becomes disconnected, just one more step of the RAES process is sufficient to rebuild a connected graph with good expansion properties.

In Figure 5.2 it is also interesting to notice the unimodal trend of the spectral gap as a function of the edge-failure probability: it decreases for p from 0 to 0.1 and it increases for $p > 0.1$. This indicates that the snapshots of *highly-dynamic* graphs, in which nodes are forced to frequently regenerate their neighborhoods, are better expanders than the snapshots of less dynamic graphs, in which the connections between nodes are more stable.

5.3.3 Flooding Time Analysis

We here present the results of the simulations of the flooding process (see Section 5.2.3) on the E-RAES model (see Section 5.2.1). The simulation proceeds as follows: Starting from the empty graph, we wait for the first round t_0 in which the dynamic graph $\{G_t = (V, E_t) : t \in \mathbb{N}\}$ meets the criterion defined in Section 5.3.1, then we pick a node $u_0 \in V$ uniformly at random, we simulate the flooding process with initiator u_0, and we measure the number of rounds until the flooding is complete.

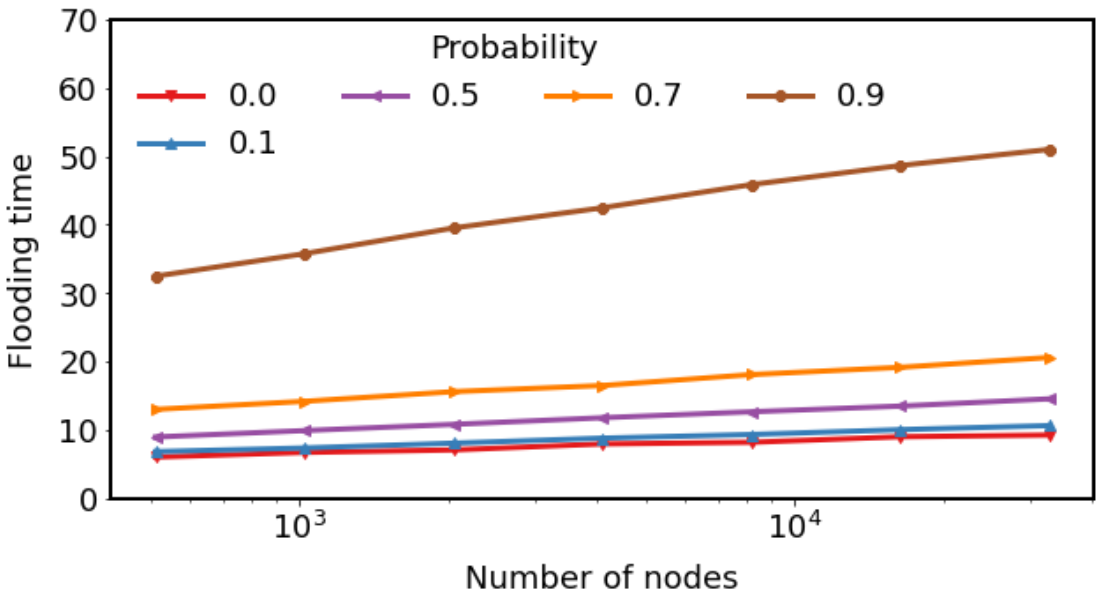


FIGURE 5.3: Semi-log-plot of the average flooding time of $\mathcal{G}(n, 4, 1.5, p)$ with $2^9 \leq n \leq 2^{15}$, $p \leq 0.9$.

Figure 5.3 shows the results of the simulations obtained by setting in the E-RAES model the parameters $d = 4$, $c = 1.5$ and different values for number of nodes n and edge-failure

probability p. Each point in the plot is the average, over 100 runs, of the number of rounds required by the flooding process to complete.

The picture quite clearly highlights that the flooding time, as a function of the number of nodes, is compatible with a logarithmic growth, for every value of the edge-failure probability p. The value of p seems to determine the multiplicative constant of the logarithm. We remark that in the simulations the message-passing step of the flooding process is scheduled *after* the edge-failure step of the E-RAES model, i.e., when for values of p larger than 0.1 the snapshot of the graph is typically disconnected. Thus it is interesting to notice that, even for large value of p, e.g. when 90% of the edges disappear at each round, the time required to get all nodes informed is quite short. These results suggest that a new message rapidly "floods" the dynamic network even if every snapshot of the dynamic graph is completely sparse and disconnected.

5.4 V-RAES simulations

In this section, we present the results of the simulations of the V-RAES model. As for the structural properties, we recall that (see Section 5.2.2) when new nodes arrive they can connect to nodes currently in the graph, but they cannot be asked for connections from other nodes. At each round thus the snapshot of the evolving graph is formed by a *core*, i.e., the nodes that were present in the graph in the previous round as well, and a *periphery*, i.e., the nodes arrived in the current round, that are connected only to nodes in the core. Hence, the snapshots of the evolving graph are not good expanders. Nevertheless, our simulations show that the "flooding time" in the V-RAES model is fast.

The definition of "flooding time" as described in Section 5.2.3 needs to be appropriately adapted in the V-RAES model to take into account the fact that new nodes join the network at any round and thus the process could (and typically does) never reach a state in which all nodes currently in the network are informed.

As we did for the E-RAES model, we want to start simulating the flooding process when the network evolution is "stationary". In Section 5.4.1 we thus define an heuristic convergence criterion and in Section 5.4.2 we present the results on the flooding process. We remark that we here present the results only for some representative parameters d and c, other choices for those parameters produce similar results.

5.4.1 Convergence criterion

Since in the V-RAES model new nodes arrive at any round with rate λ and each node leaves the network with probability q, the stationary expected number of nodes in the network is λ/q and the actual number of nodes is concentrated around its expected value. We thus consider the network evolution for the V-RAES model to have reached a stationary regime when the number of nodes in the network is close to λ/q.

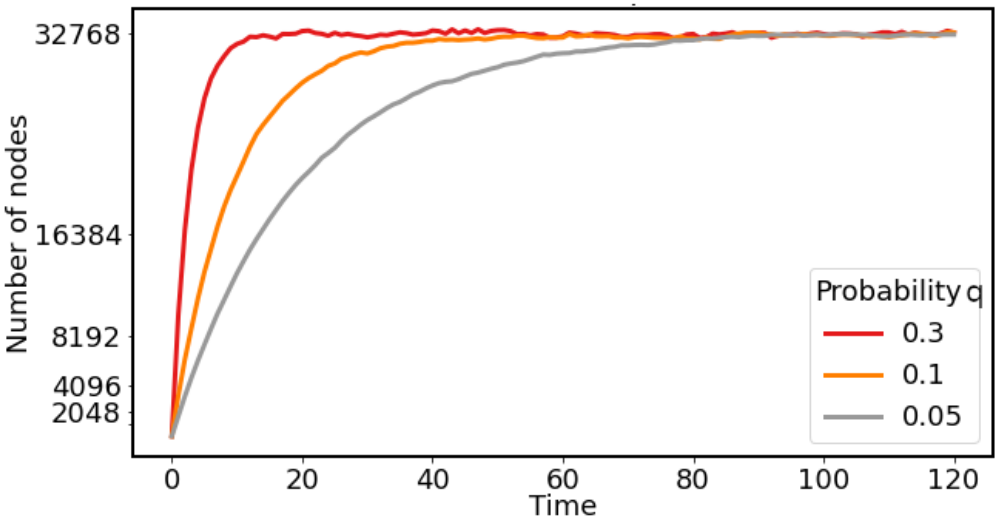


FIGURE 5.4: The evolution of the number of nodes for some sample runs with $d = 4$, $c = 1.5$, and $\lambda/q = 2^{15}$ and $q = 0.05, 0.1, 0.3$

Figure 5.4 shows the evolution of the number of nodes in the graph during the first rounds of the V-RAES, with parameters $d = 4$ and $c = 1.5$, starting from the empty graph. All plots in the picture refer to the ratio $\lambda/q = 2^{15}$, each plot with a different value for the node-leaving probability q (and with the corresponding value for λ). The number of nodes is considered *before* the node-leaving step.

5.4.2 Flooding Time Analysis

Since nodes join and leave the network at any round, in the V-RAES model a message sent from an initiator node might not reach neither all the nodes in the graph nor a large fraction of them. For example, if the initiator node and all its neighbors leave the network one round after the message departure, then the message will never reach any of the other nodes. In order to measure the speed of information spreading in the V-RAES model, we thus run the simulations as follows: Starting from the empty graph, we wait for the first round t_0 in which the dynamic graph $\{G_t = (V_t, E_t) : t \in \mathbb{N}\}$ meets the criterion defined in Section 5.4.1, then we pick a node $u_0 \in V_{t_0}$ uniformly at random,

we simulate the flooding process (see Section 5.2) with initiator u_0, and we monitor the fraction of informed nodes $\alpha_t := |I_t|/|V_t|$ at each round.

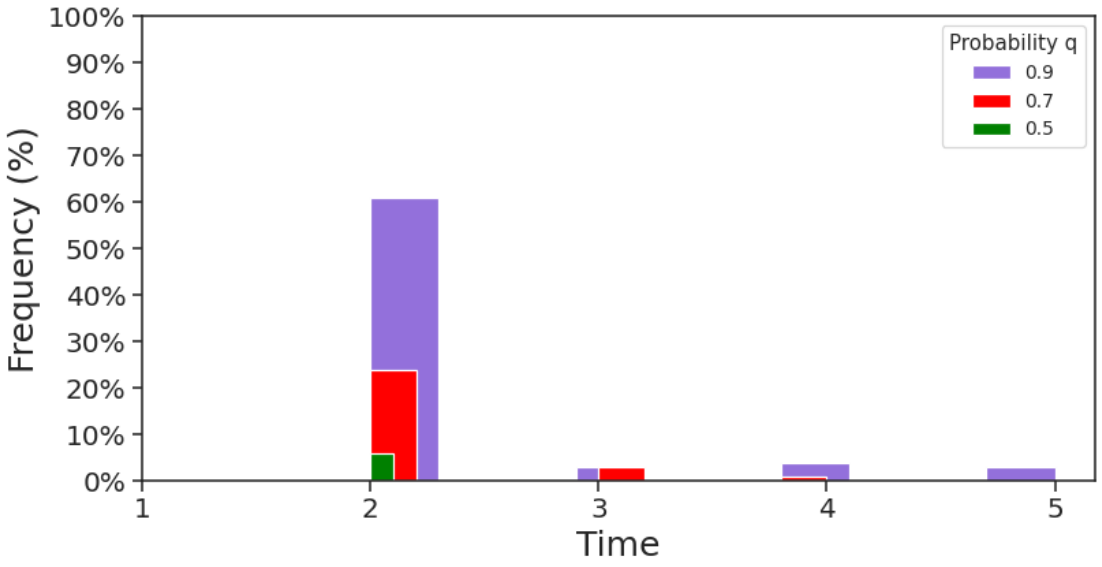


FIGURE 5.5: Percentage of the failed flooding executions. Each bar of the histogram indicates the number of times in which all the informed nodes left the network at the corresponding round. The ratio λ/q is fixed to 2^{15}.

Fig. 5.5 shows the fraction of simulations in which, at some round after t_0, all the informed nodes disappeared simultaneously, thus leaving the network without any informed node. The first observation emerging from the histograms is that all the times this event happened, it was within five rounds from t_0.

For $q = 0.9$, i.e. when about 90% of the nodes disappear at every round, in about 60% of the simulations all the informed nodes left at the second round of the flooding process. On the other hand, for $q = 0.5$, i.e when about half of the nodes disappear at every round, the fraction of times in which the message of the initiator node u_0 fails to spread in the network is very small.

In Fig. 5.6 we plot the evolution of the fraction α_t of informed nodes, for all the simulations in which the message of the initiator node u_0 does spread in the network. The plots show that, when the set of informed nodes do not disappear during the very first rounds, the fraction of informed nodes quickly stabilizes over precise values that depend on the node-leaving probability q: for $q \leqslant 0.7$ the number of informed nodes reaches a stationary phase in which almost all the nodes in the network are informed; even for larger values of the node-leaving probability, e.g., when $q = 0.9$, in all simulations in which the informed nodes do not simultaneously disappear within the first five rounds, the fraction of informed nodes stabilizes around 80%.

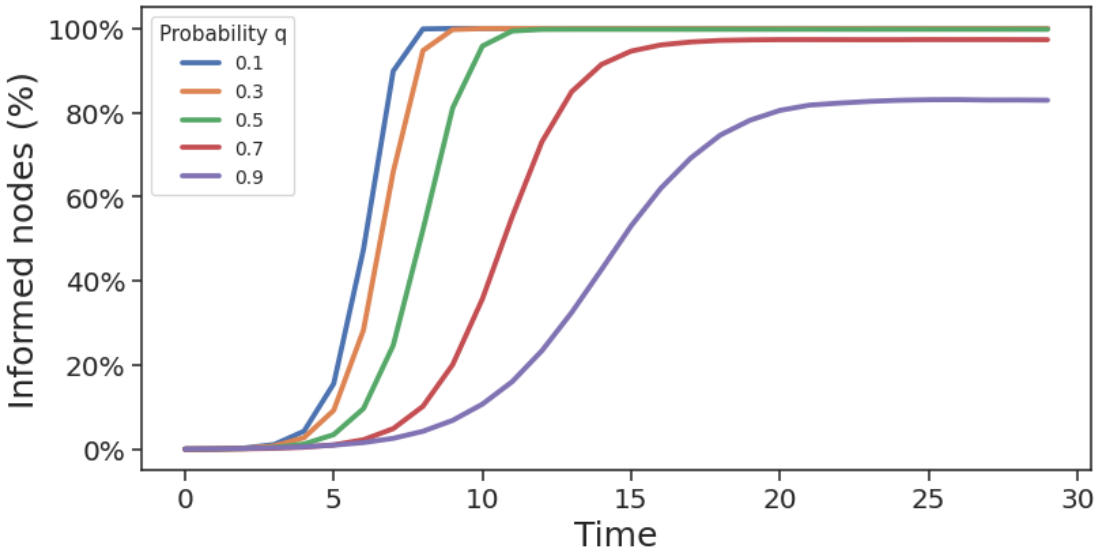


FIGURE 5.6: Average over 100 runs of the evolution of the fraction of informed nodes α_t, at each time step. In the plots the ratio λ/q is fixed to 2^{15}.

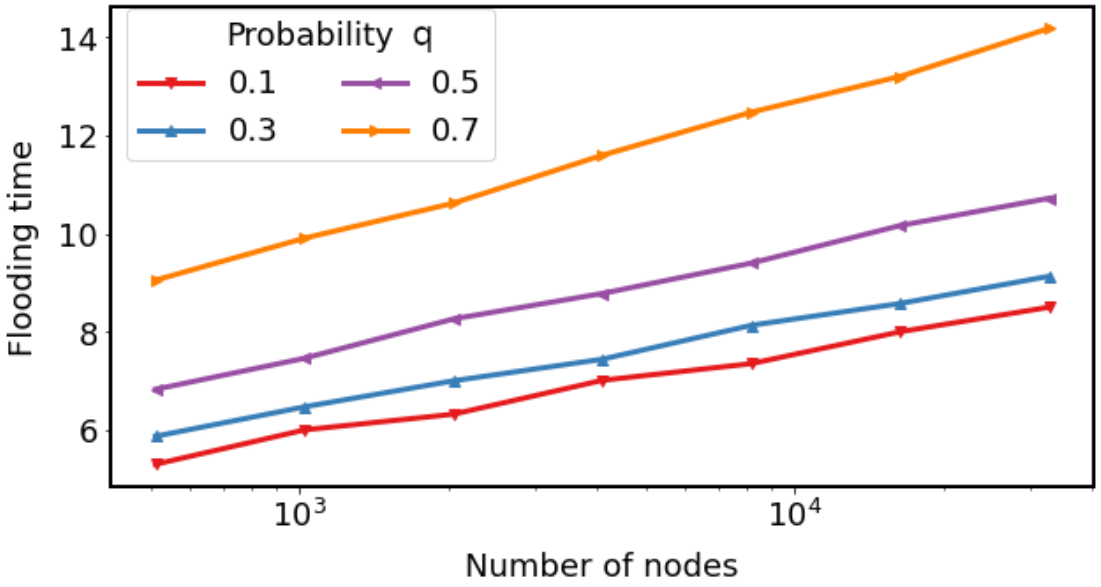


FIGURE 5.7: Semi-log-plot of the average flooding time trend of $\mathcal{G}(\lambda, q, 4, 1.5)$ with $2^9 \leq \lambda/q \leq 2^{15}$

As a measure of *flooding time* in the V-RAES model, we thus can consider the number of rounds required to reach the stable value α_t, as it is determined by the node-leaving probability q. For example, in Figure 5.7 we plot the number of rounds required by the flooding process to reach a fraction α_t of informed nodes of at least 90%, for all the values of the node-failure probability q such that the fraction of informed nodes stabilizes above 90%. The picture clearly highlights that such number of rounds is compatible with a logarithmic growth, as a function of λ/q.

5.5 EV-RAES and the parameters of the real Bitcoin networks

In this section we present a combination of the E-RAES and V-RAES models, that we call EV-RAES model, in which nodes join and leave the network as in the V-RAES and edges can be faulty as in the E-RAES. We simulate the flooding process on such a model first using the same values for d and c that we used in the V-RAES section and then using the default values of the main implementation of Bitcoin.

Starting from an arbitrary initial graph $G_0 = (V_0, E_0)$.
At each round $t \in \mathbb{N}$:

Step 0: $N_\lambda(t)$ new nodes join the graph, where $N_\lambda(t)$ is a Poisson random variable with rate λ.

Step 1: For each node u, let N_u^1 be the set of neighbors of u at the beginning of Step 1. If $|N_u^1| < d$ then u samples $d - |N_u^1|$ nodes from the set $(V_t \setminus N_\lambda(t)) \setminus N_u^1$, independently and u.a.r. with replacement, and connects to them.

Step 2: For each node u, let N_u^2 be the set of neighbors of u at the beginning of Step 2. If $|N_u^2| > c \cdot d$ then u samples $|N_u^2| - (c \cdot d)$ neighbors from the set N_u^2, independently and u.a.r. with replacement, and disconnects from them.

Step 3: Each edge $\{u, v\}$ currently in the graph disappears with probability p, independently of the other edges.

Step 4: Each node u disappears with probability q, independently of the other nodes, together with its incident edges.

We first observe the impact of the edge failures on the fraction of nodes reached in the flooding procedure and on the flooding time.

Figure 5.10 shows the results of the simulations on the fraction of informed nodes and the flooding time for the EV-RAES model, with the same values for d and c used in Section 5.4 in the simulations of the V-RAES. A comparison of Figure 5.10 with Figures 5.6 and 5.7 highlights that the impact of the edge failures on the final fraction of informed nodes and on the flooding time is quite negligible. For example, for node-leaving probability q up to 0.3, even with edge-disappearance rate $p = 0.3$ all nodes receive the message within the same amount of rounds needed when the edges do not disappear. For larger values of q, e.g. $q = 0.5$, the fraction of nodes that receive the

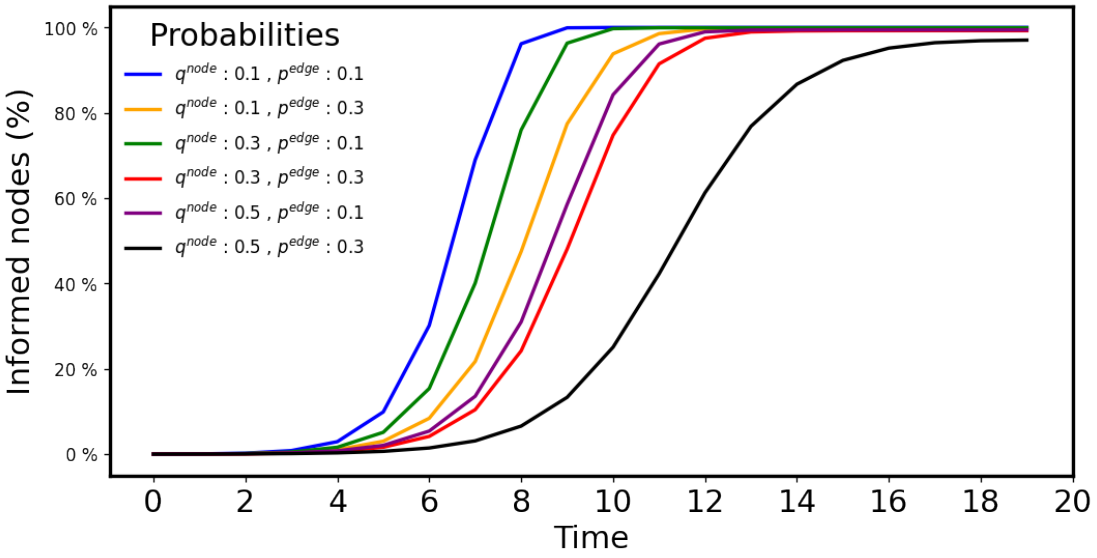


FIGURE 5.8: Evolution of the fraction of informed nodes α_t. The ratio λ/q is fixed to 2^{15}.

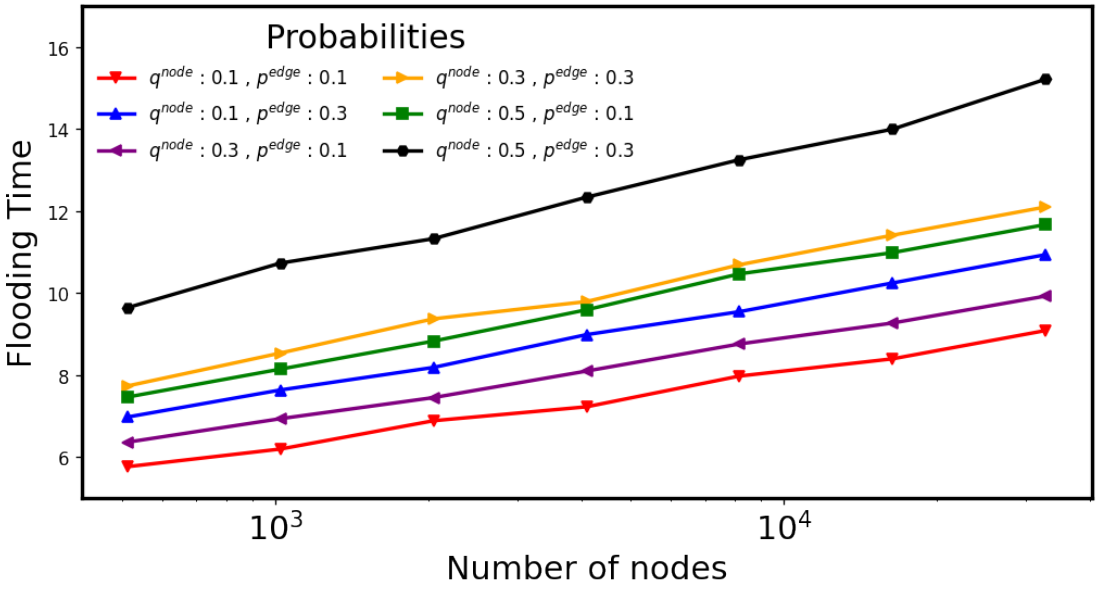


FIGURE 5.9: Semi-log-plot of the average flooding time of the EV-RAES with $2^9 \leq \lambda/q \leq 2^{15}$, node disappearance rate $q = 0.1, 0.3, 0.5$, and edge disappearance rate $p = 0.1, 0.3$.

FIGURE 5.10: EV-RAES with $d = 4$ and $c = 1.5$, fraction of informed nodes and flooding time

message turns out smaller for $p = 0.3$ with respect to the case in which edges do not disappear. Notice that such large values for q and p are only useful to test the limits of model, since they generate dynamic networks in which 50% of the nodes join and leave the network and 30% of the edges disappear at every round. In any realistic scenario, the fraction of nodes that join and leave the network at any round and the number of

connections that fail is likely to be much smaller. In those scenarios, our simulations indicate that all nodes receive the message, within a number of rounds that is compatible with a logarithmic growth as a function of the number of nodes in the network.

5.5.1 The degree of the full-nodes and network traffic

In the Bitcoin network currently there are approximately $14 \cdot 10^3$ reachable nodes (see https://bitnodes.io/ for periodic crawls of the Bitcoin P2P Network) and several hidden ones [113]. In the default configuration of the main implementation, lower and upper bounds on the number of connections that a full-node can have are set to 8 and 128, respectively. We thus also simulated the EV-RAES model with values for parameters d and c corresponding to the above values of the real Bitcoin P2P Network: namely, $\lambda/q = 2^{14}$ and $d = 8$ and $c = 15.625$.

On the one hand, a comparison of Figures 5.13 and 5.10 shows that the advantage of having such a large number of neighbors, i.e. up to 128 in Figure 5.13 as opposed to up to 6 in Figure 5.10, is limited in terms of fraction of informed nodes and flooding time. On the other hand, the number of neighbors of a full-node is directly proportional to the amount of network traffic going through the node. Indeed, "*it's common for full nodes on high-speed connections to use* 200 *gigabytes upload or more a month*" (see https://bitcoin.org/en/full-node#minimum-requirements). In order to measure the impact of the number of neighbors on the network traffic, we installed a Bitcoin-core full-node, we reduced the default number of connections of the node from 125 to 25 and, after the completion of the initial block download, we monitored the upload network traffic observing an average upload traffic between 400 and 500 MB per day, hence less than 15 GB per month.

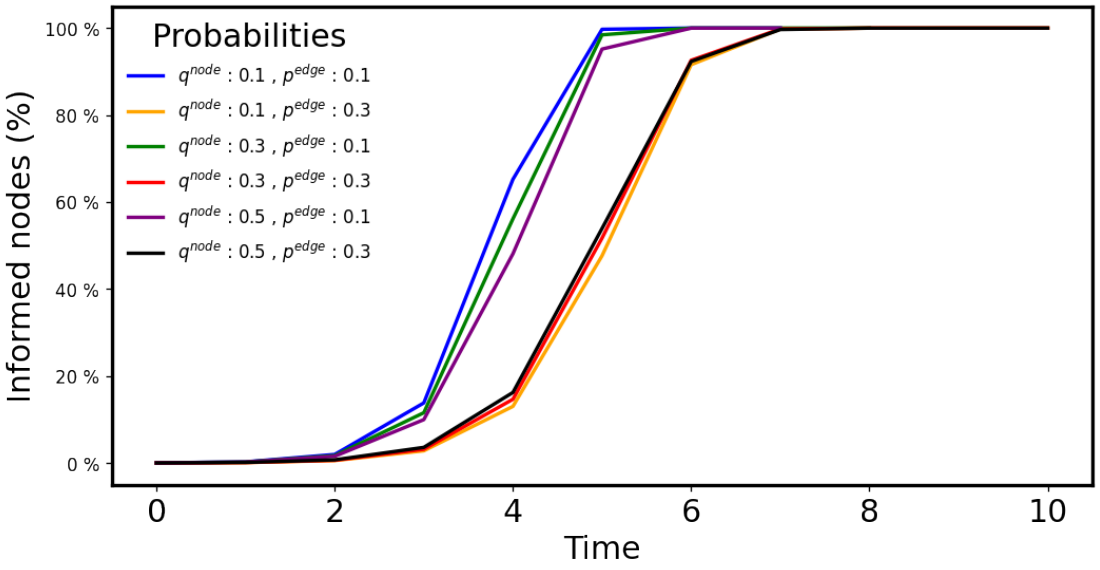


FIGURE 5.11: Evolution of the fraction of informed nodes α_t. The ratio λ/q is fixed to 2^{15}.

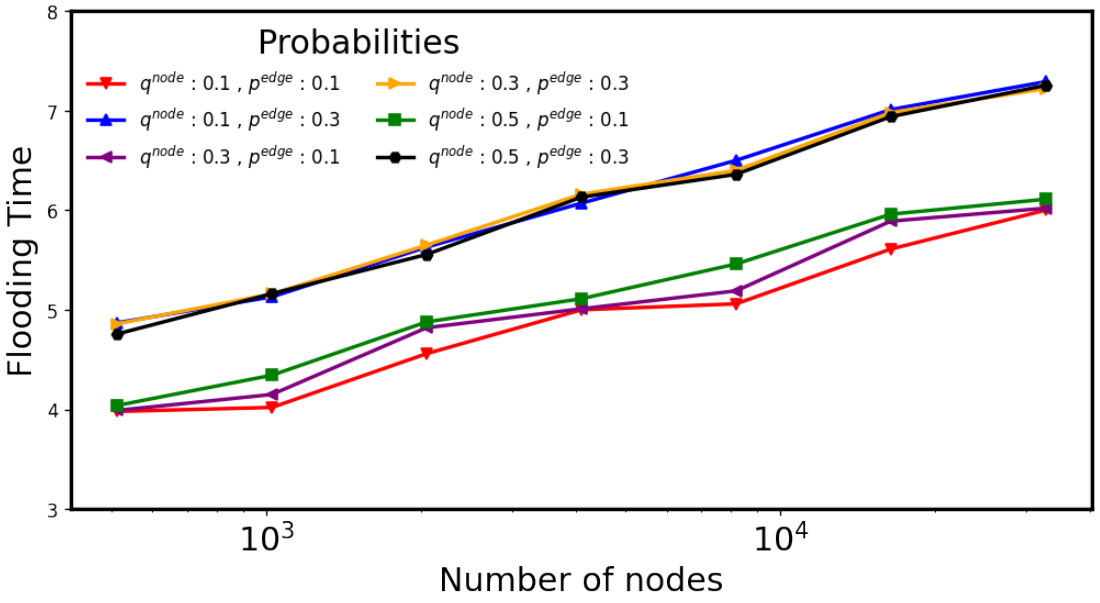


FIGURE 5.12: Semi-log-plot of the average flooding time of the EV-RAES with $2^9 \leq \lambda/q \leq 2^{15}$, node disappearance rate $q = 0.1, 0.3, 0.5$, and edge disappearance rate $p = 0.1, 0.3$.

FIGURE 5.13: EV-RAES with $d = 8$ and $c = 15.625$: Fraction of informed nodes and flooding time

Chapter 6

Highly Dynamic and Fully Distributed Data Structures

In this chapter we study robust and efficient distributed algorithms for building and maintaining distributed data structures in dynamic Peer-to-Peer (P2P) networks. We present a novel algorithm that builds and maintains with high probability a skip list for $poly(n)$ rounds despite $\mathcal{O}(n/\log n)$ churn *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). Moreover, the maintenance overhead is proportional to the churn rate. Furthermore, the algorithm is scalable in the sense 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. Our algorithm crucially relies on novel distributed and parallel algorithms to merge two n-elements 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.

6.1 Introduction

Peer-to-peer (P2P) computing is emerging as one of the key networking technologies in recent years with many application systems. These have been used to provide distributed resource sharing, storage, messaging, and content streaming, e.g., Gnutella [114], Skype [115], BitTorrent [116], ClashPlan [117], Symform [118], and Signal [119]. P2P networks are intrinsically highly dynamic networks characterized by a high degree of node *churn* i.e., nodes continuously joining and leaving the network. Connections (edges) may be added

or deleted at any time and thus the topology abruptly changes. Moreover, empirical measurements of real-world P2P networks [120–123] show that the churn rate is very high: nearly 50% of the peers in real-world networks are replaced within an hour. Interestingly, despite a large churn rate, these measurements show that the size of the network remains relatively *stable*.

P2P networks and algorithms have been proposed for a wide variety of tasks such as data storage and retrieval [124–127], collaborative filtering [128], spam detection [129], data mining [130], worm detection and suppression [131, 132], privacy protection of archived data [133], and for cloud computing services [118, 134]. Several works proposed efficient implementations of distributed data structures with low maintenance time (searching, inserting, and deleting elements) and congestion. These include different versions of distributed hash tables (DHT) like CAN [135], Chord [136], Pastry [125], and Tapestry [137]. Such distributed data structures have good load-balancing properties but offer no control over where the data is stored. Also, these show partial resilience to node failures.

To deal with more structured data in P2P networks several distributed data structures have been developed such as Skip Graphs (Aspnes and Shah [7]), SkipNets (Harvey et al., [8]), Rainbow Skip graphs (Goodrich et al., [9]), and Skip$^+$ (Jacob et al., [10]). They 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 can be seen as a major bottleneck in the implementation and use of data structures for P2P systems. Furthermore, several works deal with the problem of the maintenance of a specific graph topology [11–14], solve the agreement problem [15], elect a leader [16], and storage and search of data [17] under adversarial churn. Unfortunately, these structures are not conducive for efficient searching and querying.

In this chapter, we take a step towards designing provably robust and scalable distributed data structures and concomitant algorithms for large-scale dynamic P2P networks. More precisely, we focus on the fundamental problem of maintaining a distributed skip list data structure in P2P networks. Many distributed implementations of data structures inspired by skip lists have been proposed to deal with nodes leaving and joining the network. Unfortunately, a common major drawback among all these approaches is the lack of provable resilience against heavy churn. The problem is especially challenging since the goal is to guarantee that, under a high churn rate, the data structure must (i) be able to preserve its overall structure (ii) quickly update the structure after insertions/deletions, and (iii) correctly answer queries. In such a highly dynamic setting, it

is non-trivial to even guarantee that a query can "go through" the skip list, the churn can simply remove a large fraction of nodes in just one time-step and stop or block the query. On the other hand, it is prohibitively expensive to rebuild the data structure from scratch whenever large number of nodes leave and a new set of nodes join. Thus we are faced with the additional challenge of ensuring that the maintenance overhead is proportional to the number of nodes that leave/join. In a nutshell, our goal is to design and implement distributed data structures that are resilient to heavy adversarial churn without compromising simplicity or scalability.

6.1.1 Model: Dynamic Networks with Churn

Before we formally state our main result, we discuss our dynamic network with churn (DNC) model, which is used in previous works to model peer-to-peer networks in which nodes can be added and deleted at each round by an adversary (see e.g. [11, 96]).

We consider a synchronous dynamic network controlled by an *oblivious* adversary, i.e., the adversary does not know the random choices made by the nodes. The adversary fixes a dynamically changing sequence of sets of network nodes $\mathcal{V} = (V_0, V_1, V_2, \dots, V_t, \dots)$ where $V_t \subset U$, for some universe of nodes U and $t \geq 1$, denotes the set of nodes present in the network during round t. A node u such that $u \in V_t$ and $u \notin V_{t+1}$ is said to be *leaving* at time $t+1$. Similarly, a node $v \notin V_t$ and $v \in V_{t+1}$ is said to be *joining* the network at time $t+1$. Each node has a unique ID and we simply use the same notation (say, u) to denote both the node as well as its ID. The lifetime of a node u is (adversarially chosen to be) a pair (s_u, t_u), where s_u refers to its *start time* and t_u refers to its *termination time*. The size of the vertex set is assumed to be stable $|V_t| = n$ for all t; this assumption can be relaxed to consider a network that can shrink and grow arbitrarily as discussed in Section 6.2.1. Each node in U is assumed to have a unique ID chosen from an ID space of size polynomial in n. Moreover, for the first $B = \beta \log n$ rounds (for a sufficiently large constant $\beta > 0$), called the *bootstrap phase*, the adversary is *silent*, i.e., there is no churn, more precisely, $V_0 = V_1 = \cdots = V_B$. We can think of the bootstrap phase as an initial period of stability during which the protocol prepares itself for a harsher form of dynamism. Subsequently, the network is said to be in *maintenance phase* during which $\mathcal{V}$ can experience churn in the sense that a large number of nodes might join and leave dynamically at each time step.

Communication is via message passing. Nodes can send messages of size $\mathcal{O}(\log n)$ bits to each other if they know their IDs, but no more than $\mathcal{O}(\text{polylog}(n))$ incoming and outgoing messages are allowed at each node per round. Furthermore, nodes can create and delete edges over which messages can be sent/received. This facilitates the creation

of structured communication networks. A bidirectional edge $e = (u, v)$ is formed when one of the end points u sends v an invitation message followed by an acceptance message from v. The edge e can be deleted when either u or v sends a delete message. Of course, e will be deleted if either u or v leaves the network.

During the maintenance phase, the adversary can apply a churn up to $\mathcal{O}(n/\log n)$ nodes per round. More precisely, for all $t \geq B$, $|V_t \setminus V_{t+1}| = |V_{t+1} \setminus V_t| \in \mathcal{O}(n/\log n)$ and the adversary is only required to ensure that any new node that joins the network must be connected to a distinct pre-existing node in the network; this is to avoid too many nodes being attached to the same node, thereby causing congestion issues.

Each node can store data items. We wish to maintain them in the form of a suitable dynamic and distributed data structure. For simplicity in exposition, we assume that each node u has one data item which is also its ID. This coupling of the node, its ID, and its data allows us to refer to them interchangeably as either node u or data item u. In Section 6.2.8, we will discuss how this coupling assumption can be relaxed to allow a node to contain multiple disparate data items.

6.1.2 Problem Statement

Our primary goal is to build and maintain a data structure of all the items/nodes in the network. The data structure takes at most I rounds to insert a data item u and at most R rounds to remove it. I and R are parameters that we would like to minimize. The distributed data structure should be able to answer membership queries. Specifically, each query $q(x, r, s)$ is initiated at some source node s in round r and asks whether data item x is present in the network currently. This implies that node s now wishes to know if some node in the network has the value x. The query should be answered by round $r + Q$ where Q is the *query time*, i.e., the time to respond to queries. If $t_s > r + Q$, i.e., node s is in the network until round $r + Q$, it must receive the response. We are required to give the guarantee that the query will be answered correctly as long as either (i) there is a node with associated value x whose effective lifetime subsumes the time range $[r, r + Q]$ (in which case the query must be answered affirmatively) or (ii) there is no node with value x whose effective lifetime has any overlap with $[r, r + Q]$ (in which case the query must be answered negatively). In all other cases, we allow queries to be answered incorrectly.

In addition, all our algorithms must satisfy a dynamic notion of *resource-competitiveness* [138]. Let T be any interval between two time instants t_s and t_e. We require the work in the interval T to be proportional (within polylog(n) factors) to the amount of churn experienced from time $t_s - \mathcal{O}(\log n)$ to t_e. Formally, define the amount of work W_t as

the overall number of exchanged messages plus newly formed edges among nodes at time t. Let $\mathcal{W}(t_s, t_e) = \sum_{i=t_s}^{t_e} W_i$ and $C(t_s, t_e)$ to be the *amount of work* and churn experienced in the interval of time between t_s and t_e, respectively. We require that $\mathcal{W}(t_s, t_e) \in \tilde{\mathcal{O}}\left(C(t_s - \mathcal{O}(\log n), t_e)\right)$ w.h.p. for any interval of time t_s, t_e, where with the notation $\tilde{\mathcal{O}}(\cdot)$ we ignore polylog(n) factors (see Figure 6.1). Formally,

Definition 6.1 (Dynamic Resource Competitiveness)**.** An algorithm $\mathcal{A}$ is (α, β)-dynamic resource competitive if for any time instants t_s and t_e such that $t_e > t_s$, $\mathcal{W}(t_s, t_e) \in \mathcal{O}(\beta C(t_s - \alpha, t_e))$.

In this work, whenever we refer to an algorithm as dynamic resource competitive, we mean that the algorithm is (α, β)-dynamic resource competitive with $\alpha = \mathcal{O}(\log n)$ and $\beta = \text{polylog}(n)$.

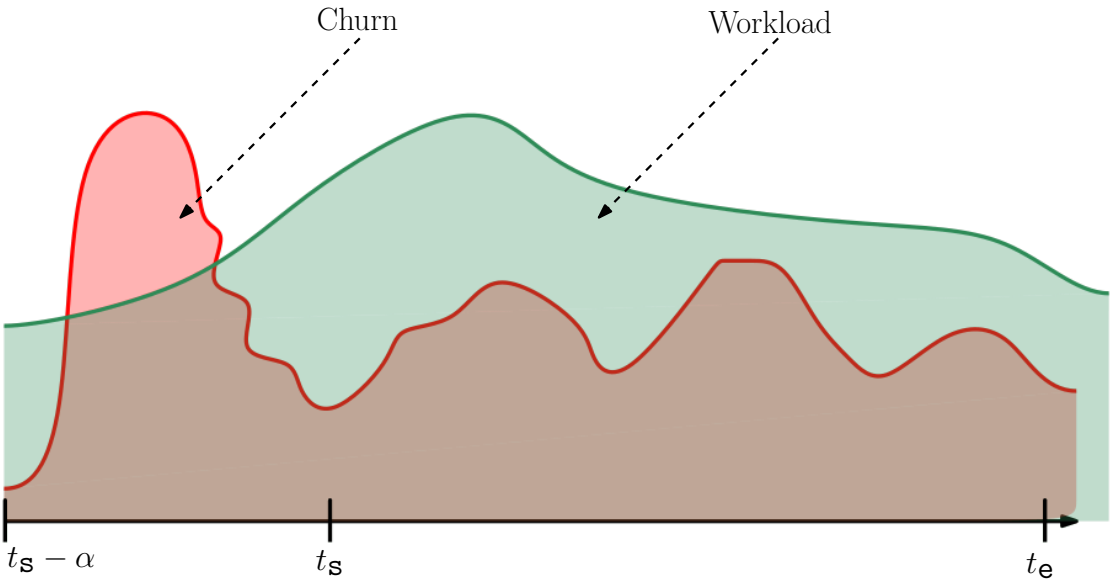


FIGURE 6.1: Visual representation of the (α, β)-dynamic resource competitive property. Red area is the experienced Churn over time. While the green area is the workload of our maintenance algorithm. The workload is proportional up to some β factor of the Churn between $t_s - \alpha$ and t_e.

6.1.3 Our Contributions

Main Result. We address the problem by presenting a rigorous theoretical framework for the construction and maintenance of distributed skip lists in highly dynamic distributed systems that can experience heavy churn.

Any query to a data structure will require some response time, which we naturally wish to minimize. Furthermore, queries will be imprecise to some extent if the data structure is dynamic. To see why, consider a membership query of the form "is node u in the

dynamic skip list?" initiated in some round r asking if a node u is in the network or not. Consider a situation where u is far away from the node at which the query was initiated. Suppose u is then churned out shortly thereafter. Although u was present at round r, it may or may not be gone when the query procedure reached u. Such ambivalences are inevitable, but we wish to limit them. Thus, we define an efficiency parameter Q such that (i) queries raised at round r are answered by round $r + Q$ and (ii) response must be correct in the sense that it must be "Yes" (resp., "No") if u was present (resp., not present) from round r to $r + Q$. If u was only present for a portion of the time between r and $r + Q$, then either of the two answers is acceptable. Quite naturally, we wish to minimize Q and for our dynamic data structure, we show that $Q \in \mathcal{O}(\log n)$.

Our algorithms ensure that the resilient skip list is maintained effectively for at least $poly(n)$ rounds with high probability (i.e., with probability $1 - 1/n^{\Omega(1)}$) even under high adversarial churn. Moreover, the overall communication and computation cost incurred by our algorithms is proportional (up to polylog(n) factors) to the churn rate, and every node sends and receives at most $\mathcal{O}(\text{polylog}(n))$ messages per round. In particular, we present the following results (the in-depth descriptions are given in Section 6.2):

1. A novel algorithm that constructs and maintains a skip list in a dynamic P2P network with an adversarial churn rate up to $\mathcal{O}(n/\log n)$ *per round*.
2. A novel distributed and parallel algorithm to merge a skip list $\mathcal{B}$ with a base skip list $\mathcal{C}$ in logarithmic time, logarithmic number of messages at every round and an overall amount of work proportional to the size of $\mathcal{B}$, i.e., to the skip list that must be merged with the base one. While this merge procedure serves as a crucial subroutine in our maintenance procedure, we believe that it is also of independent interest and could potentially find application in other contexts as well. For example, it could be used to speed up the insertion of a batch of elements in skip list-like data structures. Similarly, we designed an efficient distributed and parallel algorithm to delete a batch of elements from a skip list in logarithmic time with overhead proportional to the size of the batch.
3. A general framework that we illustrate using skip lists, but can serve as a building block for other complex distributed data structures in highly dynamic networks (see Section 6.2.7 for more details).

To the best of our knowledge, our approach is novel and fully-distributed skip list data structure that works under highly dynamic settings (high churn rates per step). Furthermore, all the proposed algorithms are localized, easy to implement and scalable. Our major contribution can be summarized in the following theorem.

Theorem 6.2 (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, we*

- *provide an* $\mathcal{O}(\log n)$ *round algorithm to construct 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,*
- *describe a fully distributed algorithm that maintains the resilient skip list with every new node inserted into the data structure in* $\mathcal{O}(\log n)$ *rounds such that membership queries 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, our algorithms are dynamically resource competitive according to Definition 6.1. The maintenance protocol ensures that the resilient skip list is maintained effectively for at least* $poly(n)$ *rounds with high probability.*

Implications. A consequence of our maintenance algorithm is that it opens up opportunities for more general distributed computation in the DNC model. In fact, this thesis formally shows how to build and maintain a non-trivial data structure under such a high dynamic settings. We now informally discuss several ideas that illustrate how our framework can be generalized. Of course, significant followup work is required to formally prove our claims. We believe this can be extended to maintaining *any* pointer-based data structure. In particular, we believe we can maintain an arbitrary *graph* over the DNC and solve a wide range of fundamental distributed graph computation tasks in the DNC model that, until now, seemed implausible.

Our current approach assumes data items are embedded into the nodes. It might be more feasible to consider the overlay network and the graph structure as two separate entities. This will allow us to retain the graph structure even when nodes in the overlay network are deleted by the adversary.

As an example of graph maintenance we briefly discuss how to build and maintain a (constant-degree) expander graph from an *arbitrary* connected graph. We will outline how this can be accomplished using an $\mathcal{O}(\log n)$ rounds maintenance cycle, which is a consequence of this work and prior works. First, during the bootstrap phase, we build our overlay maintenance network in which each node u has access to a set of well-mixed node IDs[1]. Once such a network has been built, the bootstrap phase continues with

[1]Sampling from a set of well-mixed tokens is equivalent to sampling uniformly at random from the set $\{1, \ldots, n\}$.

the constant degree expander construction. Such a task can be accomplished using (for example) the *Request a link, then Accept if Enough Space* (RAES) protocol by Becchetti et al., [6] with parameters $d \geq 1$ and $c \geq 2$. This technique builds a constant degree expander (in which all the nodes have degrees between d and $c \cdot d$) in $\mathcal{O}(\log n)$ rounds and using overall $\mathcal{O}(n)$ messages w.h.p. Once the constant degree expander is constructed, the bootstrap phase ends and the adversary begins to exert its destructive power on the overlay network. Using our data structure maintenance protocol in parallel with the expander maintenance by Augustine et al., [11] we can maintain a constant degree expander using $\mathcal{O}(\log n)$ rounds maintenance cycles in which each node sends and receives $\mathcal{O}(\text{polylog}(n))$ messages at each round.

The minimum spanning tree (MST) problem can be solved efficiently in the DNC model. In the MST problem we are given an arbitrary connected undirected graph G with edge weights, and the goal is to find the MST of G. This can be accomplished using a $\text{polylog}(n)$ rounds bootstrap phase and maintenance cycles. During the bootstrap phase, we build the overlay churn resilient network in which each node has access to a set of well-mixed tokens and we build a constant-degree expander H overlay on the given graph G (the expander edges are added to G's edge set). For this, we convert the expander (that is not addressable) into a butterfly network (that is addressable) which allows for efficient routing between any two nodes in $\mathcal{O}(\log n)$ rounds. This conversion can be accomplished using techniques of [14, 139–142]. All these protocols takes $\text{polylog}(n)$ rounds and $\tilde{\mathcal{O}}(n)$ messages to convert a constant-degree expander into an hypercubic (i.e., butterfly) network. Using the addressable butterfly on top of G, we can efficiently implement the Gallagher-Humblet-Spira (GHS) algorithm [143] as shown by Chatterjee et al. [144] to compute the MST of G in $\text{polylog}(n)$ rounds and $\tilde{\mathcal{O}}(n)$ messages using routing algorithms for hypercubic networks [145, 146]. After the bootstrap phase, we maintain (and update) the MST using our data structure maintenance protocol, the expander maintenance technique described above, and the MST computation techniques used in the bootstrap phase.

Another implication is the construction and maintenance of other more sophisticated data structures like skip graphs and its ilk [7, 9, 10]. All these data structures are not resilient to churns, and their maintenance protocols are not fast enough to recover the data structure after the failure of some nodes. Indeed, these protocols (see [7, 9, 10]) might need $\mathcal{O}(n)$ rounds to rebuild the skip-graph and they strictly require no additional churn to be able to fix the data structure. Our maintenance protocol overcomes these problems and provides a $\mathcal{O}(\log n)$ rounds skip graph constructing protocol and a technique able to repair them in $\mathcal{O}(\log n)$ in the presence of an almost linear churn of $\mathcal{O}(n/\log n)$ *at every round*. Moreover, our maintenance mechanism allows for the users to query the data structure while being maintained.

Finally, some ideas from our results could be used in the centralized batch parallel setting to quickly insert batches of new elements in skip lists (or skip graphs) data structures (see e.g.,[147]) and in the fully dynamic graph algorithms settings (see for example the survey [148]) to perform fast updates of fully dynamic data structures.

High-level Overview and Technical Contributions. Our maintenance protocol (Section 6.2) uses a combination of several techniques in a non-trivial way to construct and maintain a churn resilient data structure in polylog(n) messages per round and $\mathcal{O}(\log n)$ rounds.

Our network maintenance protocol is conceptually simple and maintains two networks– the overlay network of peers (called Spartan in Section 6.2) and the distributed data structure in which these peers can store data structure information. To ease the description of our distributed algorithm, we think of the overlay and the data structure as two different networks of degree $\mathcal{O}(\log n)$. With each peer being a part of both, the overlay and the data structure (see Figure 6.2). This way, we can think of our maintenance protocol as a collection of distributed protocols that are running in parallel and are in charge of healing and maintaining these different networks.

The overlay network maintenance protocol is conceptually similar to the ones in [14, 142]. It consists of several phases in which we ensure that the overlay network is robust to an almost linear adversarial churn of $\mathcal{O}(n/\log n)$ nodes at each round. Furthermore, the data structure maintenance protocol consists of a continuous *maintenance cycle* in which we quickly perform updates on the distributed data structure despite the high adversarial churn rate. While for maintaining the overlay network we can use the maintenance protocol in [14] as a black-box, we need to design novel algorithms to maintain the dynamic data structure.

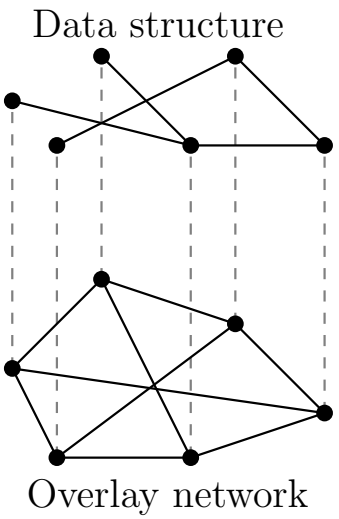


FIGURE 6.2: High-level overview.

Already existing techniques for skip lists [7, 9, 10, 149] or solving the storage and search problem in the DNC model [17, 150] can neither be used nor adapted for our purpose. We need to design novel fast distributed and parallel update protocols that are resilient to high churn rate without compromising the integrity of the data structure.

Before delving into the protocol's description, we highlight one of the key ideas of our paper. As previously mentioned, the network comprises two networks – the overlay and

the distributed data structure. However, instead of having only one network for the current skip list data structure we keep *three*: the first one is a *live* network on which queries are executed, the second one is a *clean* version of the live network on which we perform updates after every churn and the latter, is an additional temporary *buffer* network on which we store the newly added elements. Such a buffer will be promptly *merged* to the clean network during each cycle. Such a three-network architecture allows us to describe the maintenance process in a clear and simple way. (Note that the nodes are not actually replicated into multiple copies for each of the networks. The same set of nodes will maintain all the networks logically.)

Another key aspect of our protocol is that when the oblivious adversary removes some nodes from the overlay network (thus from the data structures as well), we have that (1) the overlay network maintenance protocol in [14] ensures that it will remain connected with high probability at every round and (2) all the elements that disappeared from the data structures will be *temporarily replaced* by some surviving group of nodes in the overlay network. Moreover, when (2) happens, all these replacement nodes, in addition to answering their own queries, they also answer the ones addressed to the nodes they are covering for. This is possible since we can always assume some level of redundancy in the overlay network. For example, we can assume that a node u in the overlay network has an updated copy of all its neighbors' values and pointers in the data structure. (Recall that both network are designed to be low-degree networks).

We now describe a cycle of our maintenance protocol which consists of four major phases. Moreover, we can assume that at the beginning of the cycle, live and clean networks contain the same elements. Next, assume that the adversary replaced $\mathcal{O}(n/\log n)$ from the network, this means that $\mathcal{O}(n/\log n)$ nodes have been removed from the networks (overlay, live and clean) and new $\mathcal{O}(n/\log n)$ have been added to the overlay network. After such a churn, there are some (old) nodes in the overlay network covering for the removed ones in both clean and live networks.

In the first phase, we quickly remove from the clean network those nodes that have been covered by other surviving nodes in the overlay network. In Section 6.2.3, we provide an $\mathcal{O}(\log n)$ rounds protocol that successfully polish the clean network.

In the second phase, all the newly added elements are gathered together, sorted and used to create a new temporary skip list, i.e., the buffer network. It is non-trivial to accomplish this under the DNC model in $\mathcal{O}(\log n)$ rounds. The first major step in this phase is to efficiently create a sorted list of the newly added elements. To this end we show a technique that first builds a specific type of sorting network [151], creates the sorted list, and, from that, builds the skip list in $\mathcal{O}(\log n)$ rounds. This contribution,

that can be of independent interest, shows how one can use sorting networks [151, 152] to efficiently build a skip list despite a high adversarial churn (Section 6.2.4).

In the third phase, the buffer network is merged with the clean one. To this end, we propose a novel distributed and parallel algorithm to merge two skip lists together in $\mathcal{O}(\log n)$ rounds. Intuitively, the merge protocol can be viewed as a top-down wave of buffer network nodes that is traversing the clean network. Once the wave has fully swept through the clean network, we obtain the merged skip list. All prior protocols for merging skip lists (or skip graphs) took at least $\mathcal{O}(k \cdot \log n)$ rounds to merge together two skip lists, where k is the buffer size. In our case the buffer is of size $k = \mathcal{O}(n/\log n)$ thus these algorithms would require $\mathcal{O}(n)$ rounds to perform such a merge. This is the first distributed and parallel algorithm that merges a skip list of n elements into another one in $\mathcal{O}(\log n)$ rounds. This contribution is of independent interest, we believe that the merge procedure will play a key role in extending our work to more general data structures.

In the fourth and last phase, we *update* the live network with the clean one by running a $\mathcal{O}(1)$ round protocol that applies a *local* rule on each node in the live and clean network (Section 6.2.6).

The above maintenance cycle maintains the distributed data structure with probability at least $1 - \frac{1}{n^c}$ for some arbitrarily big constant $c \geq 1$. This implies that the expected number of cycles we have to wait before getting the first *failure* is n^c and that the probability that our protocol correctly maintains the data structure for some $r < c$ rounds is at least $1 - \frac{1}{n}$.

Finally, we show how to extend our maintenance cycle to skip graphs-like data structures (Section 6.2.7) and that our technique can be easily generalized to handle cases in which each node in the overlay network possesses more than one element in the data structure (Section 6.2.8).

6.1.4 Related Works

There has been a significant prior work in designing peer-to-peer (P2P) networks that can be efficiently maintained (e.g. see [7, 125, 153–157]). A standard approach to design a distributed data structure that is provably robust to a large number of faults is to define an underlying network with good structural properties (e.g., expansion, low diameter, etc.) and efficient distributed algorithms able to quickly restore the network and data structure after a certain amount of nodes or edges have been adversarially (or randomly) removed (e.g., see [158–161]). Most prior works develop algorithms that will

work under the assumption that the network will eventually stabilize and stop changing or that an overall (somehow) limited amount of faults can occur.

Distributed Hash Tables (DHTs) (see for example [136, 137, 162–166]) are perhaps the most common distributed data structures used in P2P networks. A DHT scheme [167] creates a fully decentralized index that maps data items to peers and allows a peer to search for an item efficiently without performing flooding. Although DHT schemes have excellent congestion properties, these structures do not allow for non-trivial queries on ordered data such as nearest-neighbor searching, string prefix searching, or range queries.

To this end, Pugh [149] in the 90's introduced the skip list, a randomized balanced tree data structure that allows for quickly searching ordered data in a network. Skip lists have been extensively studied [168–171] and used to speed up computation in centralized, (batch) parallel and distributed settings [147, 172–178]. However, classical skip lists especially when implemented on a distributed system do not deal with the chance of having failures due to peers (elements) abruptly leaving the network (a common feature in P2P networks).

With the intent of overcoming such a problem, Aspnes and Shah [7] presented a distributed data structure, called a *skip graph* for searching ordered data in a P2P network, based on the *skip list* data structure [149]. Surprisingly, in the same year, Harvey et al. [8] independently presented a similar data structure, which they called SkipNet. Subsequently, Aspnes and Wieder [179] showed that skip graphs have $\Omega(1)$ expansion with high probability (w.h.p.). Although skip graphs enjoy such resilience property, the only way to fix the distributed data structure after some faults is either (i) use a repair mechanism that works only in the absence of new failures in the network and has a linear worst-case running time[2] [7] or (ii) rebuild the skip graph from scratch. Goodrich et al. [9] proposed the *rainbow skip graph*, an augmented skip graph that enjoys lower congestion than the skip graph. Moreover, the data structure came with a periodic failure recovery mechanism that can restore the distributed data structure even if each node fails independently with constant probability. More precisely, if k nodes have randomly failed, their repair mechanism uses $\mathcal{O}(\min(n, k \log n))$ messages over $\mathcal{O}(\log^2 n)$ rounds of message passing to adjust the distributed data structure. In the spirit of dealing with an efficient repairing mechanism, Jacob el at. [10] introduced SKIP$^+$, a self-stabilizing protocol[3] that converges to an augmented skip graph structure[4] in $\mathcal{O}(\log^2 n)$ rounds w.h.p., for any given initial graph configuration in which the nodes are weakly connected. The protocol works under the assumption that starting from the initial graph until the convergence to the target topology, no external topological changes happen to

[2] In the size of the skip graph.
[3] We refer to [180] for an in-depth description of self-stabilizing algorithms.
[4] It is augmented in the sense that it can be checked *locally* for the correct structure.

the network. Moreover, once the desired configuration is reached, SKIP$^+$can handle a *single* join or leave event (i.e., a new node connects to an arbitrary node in the system or a node leaves without prior notice) with a polylogarithmic number of rounds and messages. While it is shown that these data structures can tolerate node failures, there is no clarity on how to handle persistent churn wherein nodes can continuously join and leave, which is an inherent feature of P2P networks. Moreover, all the proposed repairing mechanisms [7, 9, 10, 179] will not work in a highly dynamic setting with *large, continuous, adversarial* churn (controlled by a powerful adversary that has full control of the network topology, including full knowledge and control of what nodes join and leave and at what time and has unlimited computational power).

6.2 Solution Architecture

Before delving into the description of our maintenance algorithm, we briefly introduce some notation used in what follows.

Let us briefly recall *skip lists* [149], that are randomized data structures organized as a tower of increasingly sparse linked lists. Level 0 of a skip list is a classical linked list of all nodes in increasing order by key/ID. For each i such that $i > 0$, each node in level $i-1$ appears in level i independently with some *fixed* probability p. The top lists act as "express lanes" that allow the sequence of nodes to be quickly traversed. Searching for a node with a particular key involves searching first in the highest level, and repeatedly dropping down a level whenever it becomes clear that the node is not in the current one. By backtracking on the search path it is possible to shows that no more than $\frac{1}{1-p}$ nodes are searched on average per level, giving an average search time of $\mathcal{O}(\log n)$ (see Appendix C.1 for useful properties of randomized skip lists). We refer to the *height* of a skip list $\mathcal{L}$ (see Figure 6.3) as the maximum h such that Level h is not empty. Given a node v in the skip list $\mathcal{L}$ with $n_{\mathcal{L}}$ elements, we use $N_{\mathcal{L}}^{(R,\ell)}(v)$ and $N_{\mathcal{L}}^{(L,\ell)}(v)$ to indicate v's right and left neighbors at level ℓ respectively. Moreover, when the direction and the levels are not specified we refer to the overall set of neighbors of a node in a skip list, formally we refer to $N_{\mathcal{L}}(v) = \bigcup_{\ell \geq 0}(N_{\mathcal{L}}^{(L,\ell)}(v) \cup N_{\mathcal{L}}^{(R,\ell)}(v))$. Furthermore, we define $\ell_{\max}^{\mathcal{L}}(v)$ for a node $v \in \mathcal{L}$ to be its maximum height in the skip list[5] and $\ell_{\max}(\mathcal{L}) = \max_{v \in \mathcal{L}} \ell_{\max}^{\mathcal{L}}(v)$ to be the overall height of the skip list.

Our dynamic distributed skip list data structure is architected using multiple "networks" (see Figure 6.4). 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.

[5] We omit the superscript when the skip list is clear from the context.

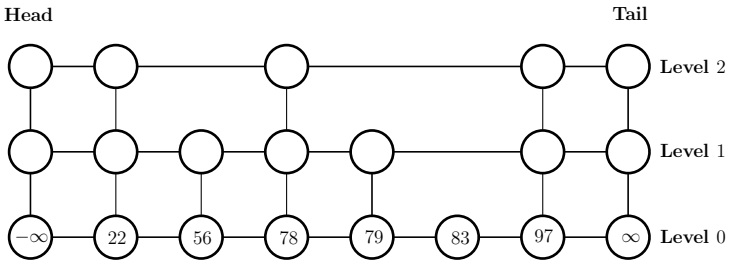


FIGURE 6.3: Skip list with $n = 6$ nodes and 3 levels.

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 [14]. 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.

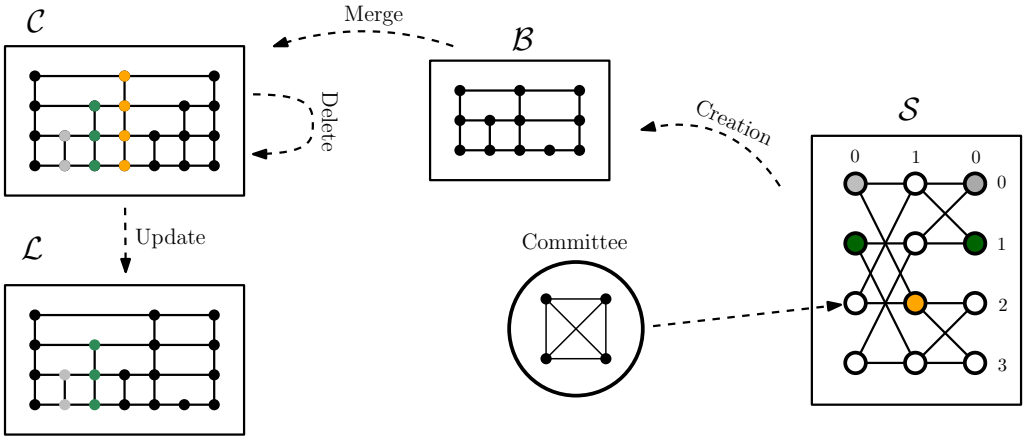


FIGURE 6.4: Schematic representation of the architecture and the Maintenance cycle described in Algorithm 4. Colored nodes in $\mathcal{L}$ and $\mathcal{C}$ are nodes that have been removed by the adversary and that are being covered by some committee of nodes (of the same color) in $\mathcal{S}$.

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[6] and it initializes the underlying network. More precisely, the bootstrap is divided in two sub-phases in which we (i) build the underlying churn resilient network described in Section 6.2.1 in $\mathcal{O}(\log n)$ rounds and, (ii) we build the skip lists data structures $\mathcal{L}$ and $\mathcal{C}$ (initially $\mathcal{L} = \mathcal{C}$) using the $\mathcal{O}(\log n)$ rounds technique described in Section 6.2.4; 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 4 is comprised of four 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 4: 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., $\mathcal{L}^{(c+1)} \leftarrow \mathcal{C}^{(c+1)}$.

6.2.1 The Spartan network

A useful technique to build and maintain a stable overlay network that is resilient to a high amount of adversarial churn is to construct and maintain a network of (small-sized) *committees* [13, 14, 16, 17]. A committee is a clique of small ($\Theta(\log n)$) size composed of essentially "random" nodes. A committee can be efficiently constructed, and more importantly, *maintained* under large churn. Moreover, a committee can be used to "delegate" nodes to perform any kind of operation. In this work, to build and maintain our churn-resilient overlay network of committees we make use of the results in [14]. Our choice is motivated by two main advantages that the Spartan network in [14] has over the previous approach [13]: (a) it tolerates an abrupt adversarial churn rate of $\mathcal{O}(n/\log n)$, and (b) it can be built in $\mathcal{O}(\log n)$ rounds with high probability. Each committee is

[6] 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.

a dynamic random clique of size $\Theta(\log n)$ in which member nodes change continuously with the guarantee that the committee has $\Theta(\log n)$ nodes as its members at any given time. These committees are arranged into a *wrapped butterfly* network [39, 181]. The wrapped butterfly has 2^k rows and k columns such that $k2^k \in \mathcal{O}(n/\log n)$ nodes and edges. The nodes correspond to committees and are represented by pairs (r, ℓ) where ℓ is the *level* or *dimension* of the committee ($0 \leq \ell \leq k$) and r is a k-bit binary number that denotes the *row* of the committee. Two committees (r, ℓ) and (r', ℓ') are linked by an edge that encodes a complete bipartite graph if and only if $\ell' = \ell + 1$ and either: (1) both committees are in the same row i.e., $r = r'$, or (2) r and r' differ in precisely the $(\ell + 1 (\text{mod } k))$th bit in their binary representation. Finally, the first and last levels of such network are merged into a single level. In particular, committee $(r, 0)$ is merged into committee (r, k). Figure 6.4 shows an example of Spartan network with 4 rows and 2 columns in which every node of the two-dimensional wrapped butterfly encodes a committee of $\Theta(\log n)$ random nodes and each edge between two committees encodes a complete bipartite graph connecting the vertices among committees. Notice that the first and last columns are the same set of committees since the butterfly is wrapped. The Spartan network is built during the bootstrap phase in which the adversary does not perform any move (having such phase is a common assumption in the DNC model [11, 14–17, 96]). In [14], the authors showed how to efficiently build such wrapped butterfly of committees in $\mathcal{O}(\log n)$ rounds using $\mathcal{O}(\log n)$ messages per node at every round. For the sake of completeness, we provide a high-level description of how to build Spartan during the bootstrap phase. As a first step, a random node v is elected as leader. Subsequently, a $\mathcal{O}(\log n)$ height binary tree rooted in v is constructed. Next, the in-order traversal number for each node is computed and a cycle between the first $N = k2^k \in \mathcal{O}(n/\log n)$ nodes is created. Each node in such a cycle becomes a committee leader. The cycle is then transformed into a wrapped butterfly network with k columns and 2^k rows. As a next step, the nodes that are not committee leaders randomly join one of the N committees with the purpose to create committee of $\Theta(\log n)$ nodes. Once such committees are created, the nodes within each committee form a clique of size $\Theta(\log n)$. Finally, the committee leaders exchange the IDs of the nodes in their committee with their neighbors so that bipartite overlay edges can be formed between nodes in neighboring committees.

After the bootstrap phase, the nodes in Spartan run a continuous $\Theta(\log \log n)$ rounds maintenance cycle in which (1) all the nodes in the network move to another committee chosen uniformly at random among all the other committees and (2) the newly added nodes are assigned to random committees. This cycle prevents the powerful adversary to grasp sensitive information about the network structure and to replace all the nodes in the network. Augustine and Sivasubramaniam [14] showed that such a maintenance

cycle makes the Spartan network robust with high probability against an adversarial churn rate of $\mathcal{O}(n/\log n)$, i.e., with high probability no committee is disrupted by the churn applied by oblivious adversary.

As a side contribution of our work, we generalize the Spartan protocol to handle an arbitrary number of nodes in the network, i.e., we relax the assumption that the size of the overlay network is fixed at each round to some number of nodes n. This makes the protocol adaptable to highly dynamic networks, in which the number of nodes might suddenly variate. We define a protocol that enlarges and reduces the dimensionality of $\mathcal{S}$ (i.e., *reshapes* $\mathcal{S}$) when the number of nodes in the Spartan network is such that the wrapped butterfly increases (resp. decreases) by one dimension. In Appendix C.2, we provide a *reshaping protocol* that reconfigures the Spartan network according to the number of peers present in the overlay network.

6.2.2 Replacing nodes that have been removed by the adversary

We describe how to maintain the live network $\mathcal{L}$ and the clean network $\mathcal{C}$ when churn occurs. Let us assume that a node $v \in V$ left the network. To preserve $\mathcal{C}$ and $\mathcal{L}$ structures, we require its committee in $\mathcal{S}$ "to cover" for the disappeared node. In other words, when a node v leaves the network, its committee members in $\mathcal{S}$ will take care of all the operations involving v in $\mathcal{L}$ and $\mathcal{C}$. Doing so, will temporarily preserve the structure of these networks, as if v was still in the network. Moreover, we assume that at every round in the Spartan network, all the nodes within the same committee communicate their states (in $\mathcal{S}, \mathcal{L}$, and $\mathcal{C}$) to all the other committee members, i.e., each node u in a committee knows all its neighbors $N_{\mathcal{S}}(v)$, each neighbor in both skip list $\mathcal{L}$ and $\mathcal{C}$ for each $v \in N_{\mathcal{S}}(v)$, and all the committees of its neighbors in $\mathcal{L}$ and $\mathcal{C}$[7]. Furthermore, when a node v leaves the network, each node in its committee in $\mathcal{S}$ will connect to v's neighbors in $\mathcal{L}$ and $\mathcal{C}$. This can be done in constant time and will increase the degree of the nodes in the data structures to $\mathcal{O}(\text{polylog}(n))$. Notice that each committee in $\mathcal{S}$ is guaranteed to be robust against a $\mathcal{O}(n/\log n)$ churn rate, i.e., the probability that there exists some committee of size $o(\log n)$ is at least $1/n^d$ for an arbitrarily large $d \geq 1$ (see Theorem 6 in [182]). This implies that every committee of nodes in $\mathcal{S}$ is able to cover for the removed nodes in $\mathcal{L}$ and $\mathcal{C}$ with high probability. Figure 6.5 shows an example of how the committees in the Spartan network cover for its members that were removed by the adversary. More precisely, in Figure 6.5(b) we show how the orange node's committee is covering for it after being dropped by the adversary. An alternative way to see this

[7]The expected space needed to store the committee's coordinates of v' neighbors in $\mathcal{L}$ and $\mathcal{C}$ is $\mathcal{O}(\log n)$ because every node in a skip list has expected degree of $\mathcal{O}(\log n)$, w.h.p.

is to assume that the committee creates a temporary virtual red node in place of the orange one (see Figure 6.5(c)).

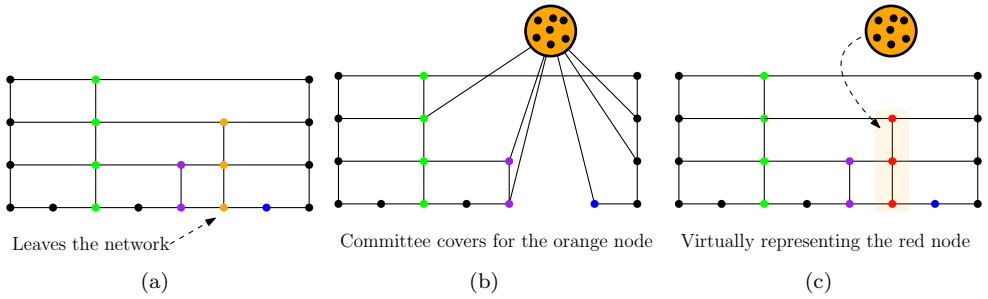


FIGURE 6.5: In Figure 6.5(a) the orange node gets removed by the adversary from the overlay network. Figure 6.5(b) shows how all its committee neighbors in the overlay cover for the orange node. Figure 6.5(c) is an equivalent visualization of Figure 6.5(b), the committee replaces the orange node with a "virtual" red node. This will temporarily preserve the skip list structure of Figure 6.5(a).

Lemma 6.3. *Replacing node(s) that have been removed by the adversary requires* $\mathcal{O}(1)$ *number of rounds and work proportional to the churn.*

Proof. When a node v leaves the network, all its committee neighbors create an edge with v neighbors in $\mathcal{L}$ and $\mathcal{C}$. This requires $\mathcal{O}(1)$ rounds and the amount of edge that are created by v's committee is $\mathcal{O}(\log^2 n)$ (at most $4 \log n$ edges for each node in the $\mathcal{O}(\log n)$-sized committee). Observe that in one cycle of $\mathcal{O}(\log n)$ rounds we can experience a churn of $\mathcal{O}(n/\log n)$ at every time instant. Thus, the overall work is $\mathcal{O}(n \cdot \text{polylog}(n))$ and respects our dynamic resource competitiveness constraints. □

6.2.3 Deletion

We describe how to safely clean the network $\mathcal{C}$ in $\mathcal{O}(\log n)$ rounds from the removed nodes for which committees in the Spartan network $\mathcal{S}$ are temporarily covering for. The elements in the clean network $\mathcal{C}$ before the cleaning procedure can be of two types: (1) not removed yet and (2) patched-up, i.e., nodes that have been removed by the adversary and for which another group of nodes in the network is covering for. A trivial approach to efficiently remove the patched-up nodes from $\mathcal{C}$, would be the one of destroying the clean network $\mathcal{C}$ by removing these nodes and then reconstructing the cleaned skip list from scratch. However, this approach does not satisfy our dynamic resource competitiveness constraint (see Definition 6.1 and Section 6.1.3); the effort we must pay to build the skip list from scratch may be much higher than the adversary's total cost. Hence, we show an approach to clean $\mathcal{C}$ that guarantees dynamic resource competitiveness. For the sake of explanation, assume that the patched-up nodes are red and the one that should remain in $\mathcal{C}$ are black. The protocol is executed in all the levels of the skip list in parallel. The

high-level idea of the protocol is to create a tree rooted in the skip list's left-topmost sentinel by backtracking from some "special" leaves located on each level of the skip list. Once the rooted tree is constructed, another backtracking phase in which we build new edges connecting black nodes separated by a contiguous list of red nodes for its leaves is executed. Figure 6.6 shows an example of the delete routine (Algorithm 5) run on all the levels of the skip list in Figure 6.6(a). Moreover, Figure 6.6(b) and Figure 6.6(c) show the execution of the delete routine at level 0. First, the tree rooted in the left-topmost sentinel is built by backtracking from the leaves (i.e., from the black nodes that have at least one red neighbor). Next, the tree is traversed again in a bottom-up fashion and green edges between black nodes separated by a list of red nodes are created (Figure 6.6(d)). Figure 6.6(e) depicts the skip list after after running Algorithm 5 in parallel on every level. Finally, Figure 6.6(f) shows the polished skip list.

Algorithm 5: Delete routine for level ℓ. See Figure 6.6 for an illustration.

Tree Formation Phase ($\mathcal{O}(\log n)$ rounds):

Let B be the set of black nodes at level ℓ with at least one red neighbor. In this step, we are required to form the shortest path tree T rooted at the left-topmost sentinel with B as its leaves. A simple bottom-up approach can be used to build this tree T in $O(\log n)$ rounds.

Message Initialization ($(\mathcal{O}(1)$ rounds):

Every $b \in B$ creates a message containing a pair of items as follows.

1. $(\dot{b}, \dot{b})$ if both its neighbors are red.

2. $(\dot{b}, b)$ if its left neighbor is red and its right neighbor is black

3. $(b, \dot{b})$ if its left neighbor is black and its right neighbor is red

Each b then sends its message to its parent in T.

Propagation Phase ($\mathcal{O}(\log n)$ rounds):

This is executed by each node u on the tree T. Node u waits to hear messages from all its children. If it has only one child, it will propagate the message to its parent in the tree. Otherwise (i.e., it has two children) it receives two messages: (w, x) from its child from below and (y, z) from its child from its right. We will show later that either x and y are both dotted or neither of them are dotted.

if *both x and y are dotted* **then**

  Introduce x to y so that they can form an edge between them at level ℓ.

Node u then propagates the message (w, z) to its parent on the tree.

Lemma 6.4. *Algorithm 5 executed on a skip list at all levels $0 \leq \ell \in \mathcal{O}(\log n)$ correctly creates an edge between every pair of black nodes that are separated by a set of red nodes at each level ℓ in $\mathcal{O}(\log n)$ rounds w.h.p.. Moreover, the total work performed by this procedure is within a polylog(n) factor of the number of red nodes.*

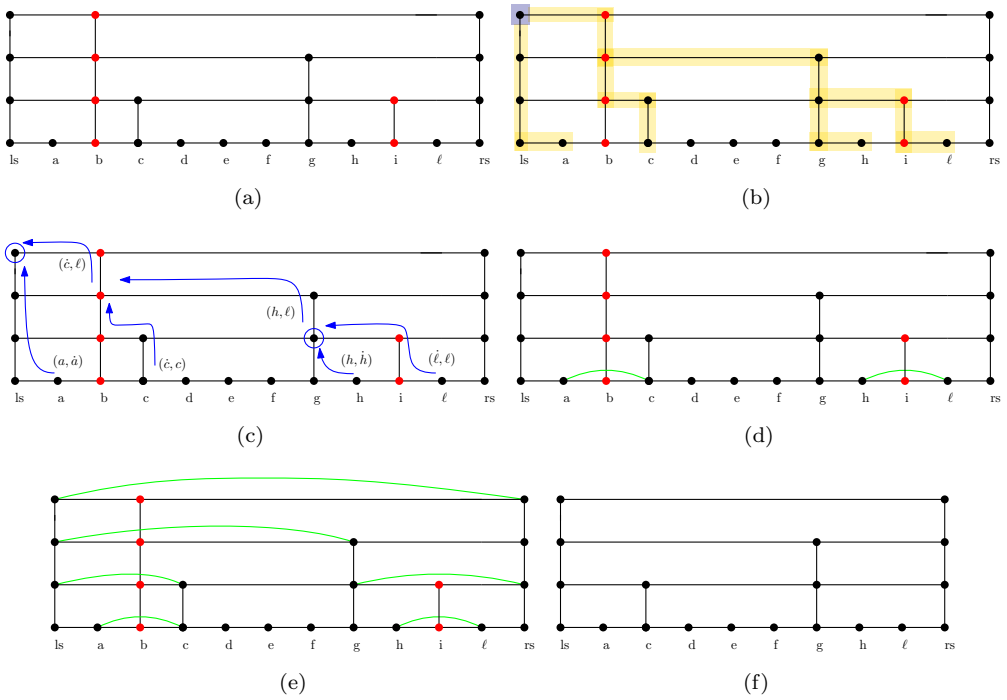


FIGURE 6.6: Example of the delete phase on a skip list. Red nodes are the one that must be deleted from the skip list, blue arrow indicates the path towards the left topmost sentinel in the skip list from the black nodes that wish wire-out its red neighbors. Figure 6.6(a) is the skip list before the deletion phase. Figure 6.6(b) shows the tree rooted in the left-topmost sentinel created by the Tree Formation Phase. Figure 6.6(c) shows the Message Initialization and the Propagation Phase executed at the bottom most level of the skip list where Blue circles indicate that an edge between two black nodes have been created. Figure 6.6(d) shows the skip list after the bottom most level run Algorithm 5. Figure 6.6(e) is the skip list after running Algorithm 5 at all its levels. Finally, Figure 6.6(f) is the skip list after the deletion phase. The left and right sentinels are represented as ls and rs, respectively.

Proof. We first establish a simple but useful invariant that holds for all nodes u in T. Consider a pair (x, y) propagated upwards by a node u to its parent. We note that x (resp., y) (in its un-dotted form) is the leftmost (resp., rightmost) leaf in the subtree rooted at u. This can be shown by induction. Clearly the statement is true if $u \in B$ as it creates the message to be of the form (u, u) (either of them possibly dotted). The inductive step holds for u at higher levels by the manner in which messages from children are combined at u.

Note that the procedure forms edges between pairs of nodes (b, b') at some level ℓ only if they are both in B (corresponding to the execution pertaining to level ℓ). Thus, for correctness, it suffices to show that the edge is formed if and only if all the nodes between b and b' at level ℓ are red.

To establish the forward direction of the bidirectional statement, let us consider the formation of an edge (b, b') at some node u that is at a level $\ell' \geq \ell$. It coalesced two

pairs of the form $(x, \dot{b})$ and $(\dot{b'}, y)$. The implication is that b has a right neighbor at level ℓ that is red and $\dot{b}$ has a left neighbor at level ℓ that is red. Moreover, they are the rightmost leaf and left most leaf respectively of the subtrees rooted at u's children. Thus, none of the nodes between b and b' at level ℓ are in T. This implies that they must all be red nodes.

To establish the reverse direction, let us consider two nodes b and b' at level ℓ that are both black with all nodes between them at level ℓ being red. Consider their lowest common ancestor u in T. Clearly, u must have two children, say, v and w. From the invariant established earlier, the message sent by v (resp., w) must be of the form $(*, \dot{b})$ (resp., $(\dot{b'}, *)$). Thus, u must introduce b and b', thereby forming the edge between them.

Since T is the shortest path rooted at the left-topmost sentinel, its height is at most $\mathcal{O}(\log n)$ w.h.p. Both the tree formation and propagation phases are bottom-up procedures that take time proportional to the height of T. Thus, the total running time is $\mathcal{O}(\log n)$.

Finally, each node in T sends at most $\mathcal{O}(1)$ messages up to its parent in T. This implies that the number of messages sent is proportional (within polylog(n) factors) to the number of leaves in T. Also, all edges added in the tree are between pairs of nodes (b, b') in B that are consecutive at level ℓ (i.e., no other node in B lies between b and b' at level ℓ). Thus, work efficiency is established. □

6.2.4 Buffer Creation

In this section, we outline the process of constructing the Buffer network $\mathcal{B}$, which comprises the nodes that have joined the network. Conceptually, to establish a skip list from an unsorted collection of distinct elements, the initial step involves creating the lowest level by organizing a sorted list. Subsequently, for each node in this list, a random experiment is conducted to construct the upper levels. In our context, we encounter an unsorted set of nodes C, which have been introduced into the Spartan network $\mathcal{S}$ through adversarial churn, and our objective is to construct a distributed skip list from it. Initially, our challenge lies in efficiently arranging these nodes to form the lowest level of the distributed data structure. The current best-known approach to sort n elements using polylog(n) incoming and outgoing messages for each node during each round requires $\mathcal{O}(\log^3 n)$ time [183]. To improve on such result, we rely on sorting networks theory (e.g. [151, 152, 181]). A sorting network is a graph topology which is carefully manufactured for sorting. More precisely, it can be viewed as a circuit-looking DAG (directed acyclic graph) with n inputs and n outputs in which each node has exactly two inputs and two outputs (i.e., two incoming edges and two outgoing edges).

A sorting network is such that for any input, the output is monotonically sorted. A sorting network is characterized by two parameters: the number of nodes resp. *size* and the *depth*. The space required to store a sorting network is determined by its size while the overall running time of the sorting algorithm is determined by its depth. Figure 6.7 shows an example of sorting network on 4 elements. One notable construction for sorting

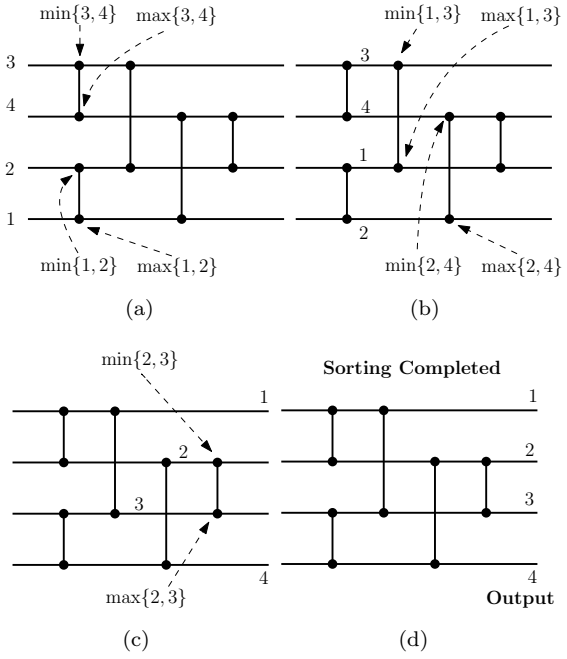


FIGURE 6.7: Example of a sorting network [152] on 4 numbers. Every crossing edge connecting two nodes in different levels is a node in the DAG representation. The upper output of a crossing edge connecting two nodes in different levels is the minimum between the two inputs, while the lower output is the maximum. Execution of the sorting network is left to right.

n elements is the $\mathcal{O}(\log n)$-depth (thus $\mathcal{O}(n \log n)$ size) AKS network, developed by Ajtai, Komlós, and Szemerédi [151] in the early 80's. However, a drawback of the AKS network is that the $\mathcal{O}(\cdot)$ notation hides large constant factors. Some progress has been made in simplifying the AKS network and improving the constant factors in its depth [184], but for practical values of n, the depth of Batcher's bitonic sort [152] remains considerably smaller. Nevertheless, if we disregard the impracticability of the AKS network, we can improve the distributed sorting result in [183] by a $\mathcal{O}(\log^2 n)$ factor. Moreover, it is well known that sorting networks can be simulated by a butterfly network and other hypercubic networks (see Chapter 3.5 in [181]). So, we first need to find a way to embed the AKS sorting network on a butterfly-like one. To this end, we use the result by Maggs and Vöcking [185] in which they show that an AKS network can be embedded on a multibutterfly [146]. A k-dimensional multibutterfly network consists of $k+1$ levels, each consisting of 2^k nodes. For $0 \leq \ell \leq k$ and $0 \leq i < 2^k$, let (ℓ, i) be the label of the ℓth node on level i. The nodes on each level ℓ are partitioned into 2^k sets $A_{\ell,0}, \ldots, A_{\ell,2^k-1}$

where $A_{\ell,j} = \{(\ell, i) : \lfloor i/2^{k-\ell} \rfloor = j\}$. The nodes in $A_{\ell,j}$ are connected to the nodes in $A_{\ell+1,2j}$ and $A_{\ell+1,2j+1}$. To embed the AKS network we need to consider a subclass of the multibutterfly networks that includes those multibutterfiles that can be constructed by superimposing butterfly networks. Furthermore, Maggs and Vöcking [185] showed how to embed the AKS network on what they called *twinbutterfly*, a multibutterfly obtained by superimposing two butterfly networks. On the twinbutterflty each leaf node has degree 4 and each non-leaf node has degree 8.

For completeness we state the result we are interested in.

Theorem 6.5 (Theorem 4.1 in [185])**.** *An AKS network $\mathcal{A}$ of size $N = \mathcal{O}(n \log n)$ can be embedded into a twinbutterfly $\mathcal{M}$ of size at most $\kappa \cdot N + o(N)$ with load 1, dilation 2, and congestion 1, where $\kappa \leq 1.352$ is a small constant depending on the AKS parameters.*

Here the *load* of an embedding is the maximum number of nodes of the AKS network mapped to any node of the twinbutterfly. The *congestion* is the maximum number of paths that use any edge in the twinbutterfly, and the *dilation* is the length of the longest path in the embedding. It follows that

Lemma 6.6. *There exists a distributed algorithm that builds a twinbutterfly network $\mathcal{M}$ in $\mathcal{O}(\log n)$ time.*

Proof. Let $n_{\mathcal{B}} = |C|$ be the number of nodes that must join together to form the Buffer network $\mathcal{B}$. The temporary twinbutterfly $\mathcal{M}$ can be built in $\mathcal{O}(\log n)$ rounds by adapting the distributed algorithm for building a butterfly network in [14] described in Section 6.2.1: (1) a leader l is elected in $\mathcal{S}$; (2) a $\mathcal{O}(\log n)$ height binary tree rooted in l is constructed; (3) a cycle using the first $n_{\mathcal{B}}(\log n_{\mathcal{B}} + 1)$ nodes of the tree in-order traversal is built; and, (4) the cycle is then transformed into the desired twinbutterfly network with $\log n_{\mathcal{B}} + 1$ columns and $n_{\mathcal{B}}$ rows in $\mathcal{O}(\log n)$ rounds. □

The idea is to use such a butterfly construction algorithm to build a twinbutterfly $\mathcal{M}$ in $\mathcal{O}(\log n)$ rounds and simulate the AKS algorithm on it. Moreover, $\mathcal{M}$ being a data structure build on top of the Spartan network, it naturally enjoys a churn resiliency property. In fact, every time a node v is churned out from $\mathcal{S}$, a committee will temporary cover for v in $\mathcal{M}$ (see Section 6.2.2). Finally, we show that the $\mathcal{B}$ can be created in $\mathcal{O}(\log n)$ rounds.

Lemma 6.7. *The Creation phase can be computed in $\mathcal{O}(\log n)$ rounds w.h.p..*

Proof. $\mathcal{M}$ can be built in $\mathcal{O}(\log n)$ rounds (Lemma 6.6). Moreover, to handle cases in which $\kappa \cdot N + o(N)$ (see Theorem 6.5) is greater than the number of nodes in the Spartan

network $\mathcal{S}$, we allow for nodes to represent a constant number of temporary "dummy" nodes for the sake of building the needed multibutterfly. Once $\mathcal{M}$ has been constructed, we run AKS on such network and after $\mathcal{O}(\log n_{\mathcal{B}}) = \mathcal{O}(\log n)$ rounds we obtain the sorted list at the base of the new skip list we want to build. Next, each node u in the buffer computes its maximum height $\ell^{\mathcal{B}}_{\max}(u)$ in the buffer skip list $\mathcal{B}$ and each level $0 < \ell \leq \ell_{\max}(\mathcal{B})$ is created by copying the base level of the skip list. Moreover, each node u at level $0 < \ell \leq \ell_{\max}(\mathcal{B})$ can be of two types, *effective* or *fill-in*. We say that a node u is *effective* at level ℓ if $\ell \leq \ell^{\mathcal{B}}_{\max}(u)$, and *fill-in* otherwise (see Figure 6.8(a)). Next, we use the same parallel $\mathcal{O}(\log n)$ rounds rewiring technique used in the Deletion phase (see Section 6.2.3) to exclude fill-in nodes from each level and obtain a skip list of effective nodes (see Figures 6.8(b)-6.8(c)). Wrapping up, to create the buffer network we need $\mathcal{O}(\log n)$ rounds to build the twibutterfly $\mathcal{M}$, $\mathcal{O}(\log n)$ rounds to run AKS on $\mathcal{M}$, $\mathcal{O}(\log n)$ rounds w.h.p. to duplicate the levels (see Lemma C.1 in Appendix C.1) and $\mathcal{O}(\log n)$ rounds to run the rewiring procedure. Thus, we need $\mathcal{O}(\log n)$ rounds[8] w.h.p. □

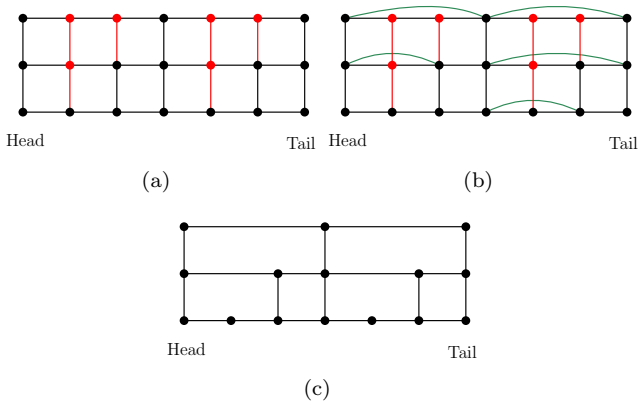


FIGURE 6.8: Example of create procedure. In Figure 6.8(a), freshly created Buffer network in which the black nodes are the effective nodes and red ones are the fill-in nodes. Figure 6.8(b) shows the buffer network after the parallel rewiring phase in which the green edges are the one created by the delete procedure. Finally, Figure 6.8(c) shows the buffer network after the Creation phase.

6.2.5 Merge

As a starting point, we analyze the process of inserting (in parallel) a sequence of sorted elements X in a skip list $\mathcal{C}$. More precisely, each element $v \in X$ starts the insertion process in $\mathcal{C}$ at the same time and from the same spot. Intuitively, all these parallel insertions can be seen as a group of elements traveling all together in the skip list. We

[8] We point out that the constant hidden by the $\mathcal{O}(\cdot)$ notation is the one for the AKS sorting network [184].

will use the term *cohesive group* to describe a group of elements traversing the skip list together. We define the *leader* of the cohesive group to be the element with the smallest ID/key among the group. In addition, the group is assumed to be small enough to ensure all nodes can exchange messages pairwise simultaneously in one round. While traversing the skip list, the leader is in charge of querying the newly encountered nodes and to communicate their ID/key to the cohesive group. Furthermore, it can happen that at some point a subset $X' \subseteq X$ separates from X and takes another route to level 0. When such a split happens, say X splits in two groups (see Proposition 6.8 below), the newly created cohesive group X' elects as a leader the element with the smallest ID/key in X'. Note that such a split requires two rounds, one for the leader to inform all nodes in the new cohesive group who their leader is and one for the nodes in the new cohesive group to "introduce" themselves to their new leader.

Proposition 6.8. *Let $X = \{x_0, x_1, \ldots, x_k\}$ where $x_{i-1} \leq x_i$ for $i \in [1, k]$ be a set of nodes that are traversing all together (i.e., a cohesive group) the skip list $\mathcal{C}$. And let $v \in X$ be a node that separates from X at some time $t \geq 0$, then all $u \in X$ such that $u \geq v$ will separate from X as well and will follow v.*

Proof. For the sake of contradiction, let $X = \{x_0, x_1, \ldots, x_k\}$ where $x_{i-1} \leq x_i$ for $i \in [1, k]$ be a cohesive group that is traversing the skip list. Moreover, assume that x_j separates from X at some time t, and that there exists x_ℓ for $\ell \geq j$ that does not follow x_j, i.e., x_ℓ does not separate from X. This generates a contradiction on the assumption that the elements in X are sorted in ascending order. Thus, it can not happen. □

Finally, we show that the time to insert a cohesive group in a skip list is at most twice the time to insert a single element in the skip list.

Lemma 6.9. *Let $X = \{x_0, x_1, \ldots, x_k\}$ where $x_{i-1} \leq x_i$ for $i \in [1, k]$ be a set of cohesive nodes that are inserted all together at the same time and from the top left sentinel node in a skip list $\mathcal{C}$. Then, the time to insert x_k along with $x_0, x_1, \ldots, x_{k-1}, x_k$ is at most twice the time to insert just x_k.*

Proof. We prove this lemma by induction. The base case of the induction is the time needed to insert x_0 and the elements before x_0 in X, i.e., $\{\emptyset\}$. Observe that the time to insert x_0 is at most twice the time to insert x_0 because it is the first element in the set X and its search path is x_0's optimal search path in $\mathcal{C}$. The inductive hypothesis is that the time to insert x_{k-1} along with $\{x_0, x_1, \ldots, x_{k-1}\}$ is at most twice the time to insert x_{k-1} and that x_{k-1}'s search path is the optimal search path when we insert only x_{k-1} from the left top-most sentinel node. As inductive step, we consider the time needed to insert x_k in $\mathcal{C}$. x_k's search path can be divided in two regions: (i) a shared

search path with x_{k-1} and (ii) an independent path in which x_k travels by itself until it reaches its destination. Moreover, observe that the shared region is a subset of the x_k's optimal search path as well, otherwise (given that x_{k-1} and x_k perform the same moves) it would contradict the inductive hypothesis about x_{k-1} search path optimality. As for the independent region, the correctness of the search/insert algorithm [149] implies that it must be a subset of the optimal search path as well. Finally, by combining the shared search path and the independent one we obtain that x_k's overall search path must be the optimal one. Given that all the nodes move in parallel, we can conclude that the time to insert x_k along $x_0, x_1, \ldots, x_{k-1}, x_k$ is at most twice the time to insert just x_k. □

WAVE: An $\mathcal{O}(\log n)$ merge procedure. Here we show one of the main contributions of this paper. We provide a distributed protocol to merge two skip lists in $\mathcal{O}(\log n)$ rounds w.h.p. Intuitively, our merge procedure can be viewed as a top-down *wave* of parallel searches. The searches are performed on $\mathcal{C}$ by elements in $\mathcal{B}$. The wave is initiated by the top-most nodes in $\mathcal{B}$. At any point in time, the wave front represents all the nodes in $\mathcal{B}$ that are actively looking for the right position in $\mathcal{C}$. As nodes on the wave front find their respective appropriate location, they get inserted into $\mathcal{C}$. Thus, when the wave has swept through $\mathcal{C}$ in its entirety, we get the required merged skip list. Moreover, we notice that there must be only one cohesive group in the top level of the skip list $\mathcal{B}$, and that each couple of cohesive groups X_ℓ, Y_ℓ at the same level $\ell < \ell_{\max}(\mathcal{B})$ must be separated by one (or more) nodes v that belongs to one (or more) cohesive group $Z_{\ell'}$ such that $\ell' > \ell$.

Proposition 6.10. *Given a skip list $\mathcal{B}$, there is only one cohesive group at $\ell_{\max}(\mathcal{B})$. Moreover, for each $0 \leq \ell < \ell_{\max}(\mathcal{B})$, let $X_\ell = \{x_0, x_1, \ldots, x_k\}$ be a cohesive group at level ℓ. Then, X_ℓ's left neighbor (i.e. x_0's left neighbor at level ℓ) is a node v such that $\ell^{\mathcal{B}}_{\max}(v) > \ell$.*

Proof. Let ℓ be the maximum height of the skip list $\mathcal{B}$ and assume to have two disjoint cohesive groups X_ℓ and Y_ℓ. This implies that X_ℓ and Y_ℓ are separated by one (or more nodes) v such that $\ell^{\mathcal{B}}_{\max}(v) > \ell$. Thus we have a contradiction on ℓ being $\mathcal{B}$'s maximum height. Next, let $0 \leq \ell < \ell_{\max}(\mathcal{B})$, consider the cohesive group $X_\ell = \{x_0, x_1, \ldots, x_k\}$ and its left neighbor v at level ℓ (i.e., $v \in N^{(L,\ell)}_{\mathcal{B}}(x_0)$). Then v must be such that $\ell^{\mathcal{B}}_{\max}(v) > \ell$. That is because, if $\ell^{\mathcal{B}}_{\max}(v) = \ell$ then we would have $v \in X_\ell$ and if $\ell^{\mathcal{B}}_{\max}(v) < \ell$, we would have $v \notin N^{(L,\ell)}_{\mathcal{B}}(x_0)$. □

Before diving into the description of the WAVE protocol, we define the left and right children of a node u at some level $\ell > 0$ in a skip list. Informally, the right (resp. left) children of a node u at level ℓ is defined by the set of *contiguous* nodes after (resp.

before) u at level ℓ in the skip list. Formally, given a node u in the buffer network $\mathcal{B}$ at level $\ell > 0$ we define $\Gamma^L_{\ell'}(u) = \{v : v < u \wedge \ell_{\max}(v) = \ell' \wedge \forall v < w < u, \ell_{\max}(w) \leq \ell'\}$and $\Gamma^R_{\ell'}(u) = \{v : v > u \wedge \ell_{\max}(v) = \ell' \wedge \forall u < w < v, \ell_{\max}(w) \leq \ell'\}$ as the left and right children of u at level $\ell' < \ell$ in $\mathcal{B}$. Moreover, we refer to u's set of children at level ℓ' as the union of its left and right ones at level ℓ', i.e., $\Gamma_{\ell'}(u) = \Gamma^L_{\ell'}(u) \cup \Gamma^R_{\ell'}(u)$ and to its overall set of children as the union of each $\Gamma_{\ell'}(u)$ for all $\ell' < \ell$, i.e., $\Gamma(u) = \bigcup_{\ell'=0}^{\ell-1} \Gamma_{\ell'}(u)$. Furthermore, we notice that two neighboring nodes u and w at level ℓ share a subset of their children. In a similar way, we introduce the concept of *parent* of a node in a skip list. Given a node u at some level $\ell < \ell_{\max}(\mathcal{B})$, we say that z is a parent of u if $u \in \Gamma(z)$. Observe that each node u such that $\ell_{\max}(u) < \ell_{\max}(\mathcal{B})$ has exactly two parents. Figure 6.9 shows an example of the children-parent relationship defined above. The children set of a node u can be seen as a depth 1 tree rooted in u where u's children are its leaves (see Figure 6.9(b)).

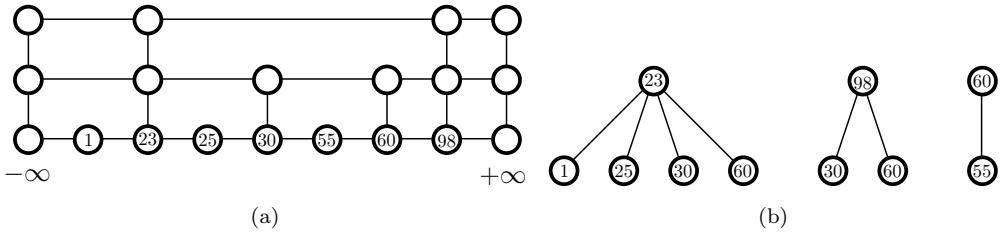


FIGURE 6.9: Example of the parent-children relationship defined above. Figure 6.9(a) is an example of a skip list. Figure 6.9(b) shows the children set Γ of nodes 23, 98, and 60, respectively.

The merge protocol is divided in two phases: (i) a preprocessing phase on the buffer network $\mathcal{B}$ and (ii) a merge phase on the clean network $\mathcal{C}$. The preprocessing phase takes care of creating the cohesive groups at each level that must be merged into $\mathcal{C}$ and to "shorten" the distance between every couple of nodes in such groups to guarantee fast communication time[9] within the cohesive groups. More precisely, let $X_\ell = \{x_0, x_1, x_2, \ldots, x_k\}$ be a set of consecutive nodes such that $\ell^{\mathcal{B}}_{\max}(x) = \ell$ for each $x \in X_\ell$ and $0 \leq \ell \leq \ell_{\max}(\mathcal{B})$. Then, each node $x_i \in X_\ell$ for $1 \leq i \leq k$ sends its ID to x_0 through its left neighbor. Doing so will allow each x_i to discover the ID of all the nodes $x_j \in X_\ell$ such that $x_j > x_i$ (see Figure 6.10). We observe that the preprocessing phase requires $\mathcal{O}(\log n)$ rounds w.h.p.

Lemma 6.11. *Given a skip list of n nodes, the preprocessing phase requires $\mathcal{O}(\log n)$ rounds w.h.p.*

The proof of Lemma 6.11 follows from the fact that each X_ℓ for $0 \leq \ell \leq \ell_{\max}(\mathcal{B})$ executes the protocol in parallel and that $|X_\ell| = \mathcal{O}(\log n)$ w.h.p. (see Lemma C.2 in

[9]After the preprocessing phase, every couple of nodes in the same cohesive group can exchange messages within $\mathcal{O}(1)$ rounds. Thus, a node $u \in \mathcal{B}$ can send messages to its children $\Gamma(u)$ in $\mathcal{O}(1)$ time as well.

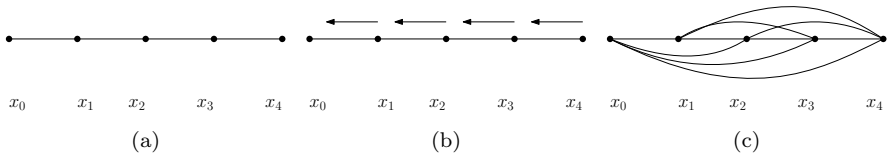


FIGURE 6.10: Example of the preprocessing operation on a cohesive group of nodes. Figure 6.10(a), Figure 6.10(b) and Figure 6.10(c) show the cohesive group before, during, and after the preprocessing phase, respectively.

Appendix C.1). Once the preprocessing phase is completed, the merge phase is executed. In such phase, each node in the buffer network $\mathcal{B}$ can be in one of the following three states {idle, merge, done}. Moreover, a node $u \in \mathcal{B}$ in idle state waits and keeps listening to its two parents in $\mathcal{B}$. A node in the merge state instead, is actively merging (as a leader or follower) itself in the clean network $\mathcal{C}$. Finally, a node that successfully merged in $\mathcal{C}$, enters the done state. Each node u in the buffer network maintains a set of children $\Gamma(u)$, parents locations (initially set to $-\infty$), a starting point for its merge (initially set to $-\infty$), and the starting level for its own merge (initially set to be the topmost level in $\mathcal{C}$, i.e., $\ell_{\max}(\mathcal{C})$). Moreover, this concept nicely extends to a cohesive group of nodes that must travel together in the skip list $\mathcal{C}$. In the first round of the merge protocol, the *unique* cohesive group $X_{\ell_{\max}(\mathcal{B})} = \{x_0, x_1, \ldots, x_k\}$ with $x_0 = -\infty$ and $x_k = +\infty$ in $\mathcal{B}$'s topmost level changes its state to merge and starts its merging phase in $\mathcal{C}$ from the left-topmost sentry. For the sake of simplicity, we consider $\mathcal{B}$'s left and right sentries as nodes of the buffer network that must be merged with $\mathcal{C}$[10]. We assume that such sentinels are contained in $\mathcal{C}$'s ones. Moreover, at each round, every cohesive group $X \subseteq \mathcal{B}$ on a node $v \in \mathcal{C}$ at level ℓ that is in the merge state performs the following steps:

1. The leader x_0 checks node v's ID/key in $\mathcal{C}$ on which X is currently located;

2. x_0 communicates v's ID/key to its followers in $\mathcal{B}$, i.e., to the nodes in $X \setminus \{x_0\}$;

3. x_0 computes the split $X' = \{u \in V : u > z\}$ where z is the right neighbor of v at level ℓ and elects the element with smallest ID/key in X' as a leader of such new cohesive group;

4. Each node $u \in \hat{X}$, where $\hat{X} = X \setminus X'$, sends a message $m(u) = \langle v, z, \text{“down”}, \ell \rangle$ to its children $\Gamma(u)$ in $\mathcal{B}$;

5. Each node $w \in X'$, sends a message $m(w) = \langle v, z, \text{“right”}, \ell \rangle$ to its children $\Gamma(w)$ in $\mathcal{B}$;

6. The new cohesive group X' separates from X and continues its merge procedure on z at level ℓ;

[10]This does not affect the merge outcome. Moreover, when $\mathcal{B}$ is completely merged with $\mathcal{C}$, we can remove $\mathcal{B}$'s sentinels from $\mathcal{C}$ in $\mathcal{O}(1)$ rounds.

7. If the current level ℓ is at most $\ell_{\max}(x_0)$, $\hat{X}$ must be merged between v and z in $\mathcal{C}$:

- **7.a.** $\hat{X}$ merges itself between v and z;
- **7.b.** Each node $u \in \hat{X}$ sends a message $m(u) = \langle u, y, \text{“down”}, \ell \rangle$ where y is u's right neighbor in the newly merged area that includes $v, \hat{X}$, and z to $\Gamma(u)$;
- **7.c.** Each node in $u \in \hat{X}$ announces its successful merge at level ℓ to its children $\Gamma(u)$;

8. If the next level $\ell - 1 \geq 0$, then $\hat{X}$ must proceed to the next level:

- **8.a.** If at least one of $\hat{X}$ neighbors at level ℓ is a node in $\mathcal{B}$, i.e., $\{v, z\} \cap \mathcal{B} \neq \emptyset$, then $\hat{X}$ waits for $\{v, z\} \cap \mathcal{B}$ to be merged at level $\ell - 1$ and proceeds with its own merge at level $\ell - 1$;

Furthermore, once a node is fully merged in level 0 of $\mathcal{C}$, it enters the done state and terminates. Next, we describe the behavior of the nodes in $\mathcal{B}$ upon receiving messages from their parents. A node v, in the buffer network $\mathcal{B}$ in idle state is continuously listening to its parents. v's behavior changes according to the type of message it is receiving. More precisely, let us assume that v receives a message of the type $m(u) = \langle L, R, \text{move}, \text{level} \rangle$ from its parent u:

If the message is sent by a node that is in the merge state at some level above $\ell_{\max}(v)$, then v has to update its pointers:

1. If $v < m(u).R$ and v.starting_point $\leq m(u).L$ then v updates its starting point and level for its own merge with the node $m(u).L$ and $m(u)$.level, respectively;
2. If $v > m(u).R$ and v.starting_point $< m(u).R$ then v updates its starting point and level for its own merge with the node $m(u).R$ and $m(u)$.level, respectively;
3. v updates its parents locations according to the move their parents performed in $\mathcal{C}$, i.e., if $m(u)$.move = “down” then it updates the location of its parent u with $m(u).L$, otherwise with $m(u).R$;
4. Checks if it can enter in the merge state, this happens if v becomes *independent* from its parents, i.e., if v.starting_point is different from the current location of both its parents in $\mathcal{C}$:
   - **4.a.** v looks at its left neighbor z at level $\ell_{\max}(v)$ in $\mathcal{B}$. If z's starting point is the same as v's one, then v sets itself as *follower*, otherwise v becomes *leader*;
   - **4.b.** If v is leader, it enters in the merge state starting the merge procedure in $\mathcal{C}$ at v.starting_point and level v.level_to_start along with its followers, i.e., the cohesive group $X = \{v\} \cup \{w : w \in N_{\mathcal{B}}^{(R,\ell_{\max}(v))}(v) \wedge v.\text{starting\_point} = w.\text{starting\_point}\}$ enters the merge state;

If v is idle and discovers that both its parents successfully merged in $\mathcal{C}$ in level $\ell = \ell_{\max}(v)$, then it must change state to merge and proceed with its own merge phase in $\mathcal{C}$ starting from v.starting_point at level v.level_to_start. Moreover, as before, it:

1. v looks at its left neighbor z at level $\ell_{\max}(v)$ in $\mathcal{B}$. If z's starting point is the same as v's one, then v sets itself as *follower*, otherwise v becomes *leader*;

2. If v is leader, it enters in the merge state starting the merge procedure in $\mathcal{C}$ at v.starting_point and level v.level_to_start along with its followers, i.e., the cohesive group $X = \{v\} \cup \{w : w \in N_{\mathcal{B}}^{(R,\ell_{\max}(v))}(v) \wedge v.\text{starting\_point} = w.\text{starting\_point}\}$ enters the merge state;

First and foremost, we notice that all the idle nodes in $\mathcal{B}$ correctly update their starting points. More precisely, a node $v \in \mathcal{B}$ that updates its starting point, it can choose only another point laying in its optimal search path from the top-left sentry to $\mathcal{C}$'s bottom most level. This property follows by the fact that each node v in $\mathcal{B}$ receives messages only by its (left and right) parents in $\mathcal{B}$, i.e., v receives a list of nodes ID that lay on the optimal search paths of its parents in the skip list $\mathcal{C}' = \mathcal{C} \cup \mathcal{B}_{>\ell}$ obtained by merging $\mathcal{C}$ and the elements of $\mathcal{B}$ situated in the level above ℓ. Thus, v updates its starting point with nodes that it would encounter had it performed a classic insertion in $\mathcal{C}'$. Thus, v's new starting point lays in its optimal search path. This property, allows the nodes in $\mathcal{B}$ to "virtually" travel in $\mathcal{C}$ using the information shared by their parents. Next, we notice that one critical part of our protocol is that a cohesive group X that was successfully merged at some level ℓ and wants to proceed one level down to $\ell - 1$ may have to wait for its parents v and z to be successfully merged in such target level. This happens if, at least one of X's neighbors at level ℓ in $\mathcal{C}$ is an element in the buffer (i.e., it is one of its parents in $\mathcal{B}$) and it is still in its merging phase at level $\ell - 1$. The next lemma provides a bound on the time a cohesive group X needs to merge itself in the skip list provided that it may have to wait for its parents to be successfully merged beforehand at each level.

Lemma 6.12. *Let X be a cohesive group at level $0 \leq \ell \leq \ell_{\max}(\mathcal{B})$ in $\mathcal{B}$. The merge time of X in $\mathcal{C}$ takes $\mathcal{O}(\log n + \ell_{\max}(\mathcal{B}) - \ell)$ rounds w.h.p.*

Proof. Assume that the cohesive group X was merged at level ℓ of the clean network and has to proceed with its own merge at level $\ell - 1$. Let v and z be X's left and right parents in $\mathcal{B}$. We have three cases to consider: (i) X is independent from its parents; (ii) X depends on at least one of its parents through all the levels $\ell' < \ell$; and, (iii) X may depend on one of its parents for some time and become independent at some point. In the first case, X is merging itself in a sub-skip list that contains only elements in the

clean network. Thus, X's overall merging time requires $\mathcal{O}(\log n)$ w.h.p. To analyze the second case, we notice that as X depends on its parents, also X's parents may depend on their parents as well and so on. We can model the time a cohesive group X at level ℓ has to wait in order to start its own merge procedure at level $\ell-1$ as the time it would have taken had it traveled in a pipelined fashion with its parents. Moreover, such pipelining effect can be extended to include, in a recursive fashion, the parents of X's parents, the parents of the parents of X's parents, and so on. Thus, X can be modeled as the tail of such "parents-of-parents" chain (or pipeline), in which X must wait for all the nodes in the pipeline to be merged at level $\ell-1$ before proceeding with its own merge in such level. Moreover, we observe that, by our assumption, X is dependent on at least one of its parents throughout the merge phase. Without loss of generality, let us assume that X depends on one parent, say z. In other words, X is never splitting, and its placing itself right after/before z at each level $\ell' < \ell$. The time X takes to merge in one level ℓ' after z is $\mathcal{O}(1)$ (see Lemma 6.9). Thus, X merge itself in some level ℓ' by time $T_z^{\ell'} + \mathcal{O}(1)$ where $T_z^{\ell'}$ is the time z needs to merge itself in level ℓ'. Furthermore, z may be dependent to one of its parents, say w. Thus the time to insert z at level ℓ' is $T_z^{\ell'} = T_w^{\ell'} + \mathcal{O}(1)$. If we repeat this argument for all the levels $\hat{\ell} > \ell'$ we obtain that the time to merge the cohesive group X at level ℓ' can be defined as $T_y^{\ell'} + \ell_{\max}(\mathcal{B}) - \ell + 1 = T_y^{\ell'} + \ell_{\max}(\mathcal{B}) - \ell_{\max}^{\mathcal{B}}(X) + 1$ where y is the topmost parent in the pipeline of dependencies. Thus X's overall merge time is $T_X^0 = T_y^0 + \ell_{\max}(\mathcal{B}) - \ell_{\max}^{\mathcal{B}}(X) + 1 = \mathcal{O}(\log n + \ell_{\max}(\mathcal{B}) - \ell_{\max}^{\mathcal{B}}(X) + 1)$ w.h.p. In the last case, X may depend on its parents for some time and then gain independency. Assume that X depends its parent z until level ℓ' and than gains independency. X's overall merge time is $T_X^0 = \mathcal{O}(T_z^{\ell'} + 1 + \log n) = \mathcal{O}(\log n)$ w.h.p. □

Given Lemma 6.12, we can bound the overall running time of the WAVE protocol.

Lemma 6.13. *Let $\mathcal{C}$ and $\mathcal{B}$ be two skip lists of n elements built using the same p-biased coin. Then, the WAVE protocol merges $\mathcal{C}$ and $\mathcal{B}$ in $\mathcal{O}(\log n)$ rounds w.h.p..*

Proof. The proof follows by noticing that a idle cohesive group X in some level ℓ starts its merge procedure when the wave sweeps through it. This happens within $\mathcal{O}(\log n)$ rounds w.h.p. from the WAVE protocol initialization time instant. Moreover, a idle node/cohesive group, after entering the merge state needs $\mathcal{O}(\log n + \ell_{\max}(\mathcal{B}) - \ell)$ rounds w.h.p. to be fully merged in $\mathcal{C}$ (Lemma 6.12). That is, the merge time is at most $\mathcal{O}(\log n + \log n + \ell_{\max}(\mathcal{B}) - \ell) = \mathcal{O}(\log n)$ w.h.p. □

Figure 6.11 shows an example of the WAVE protocol described above, each colored wave represent a round of the merging protocol.

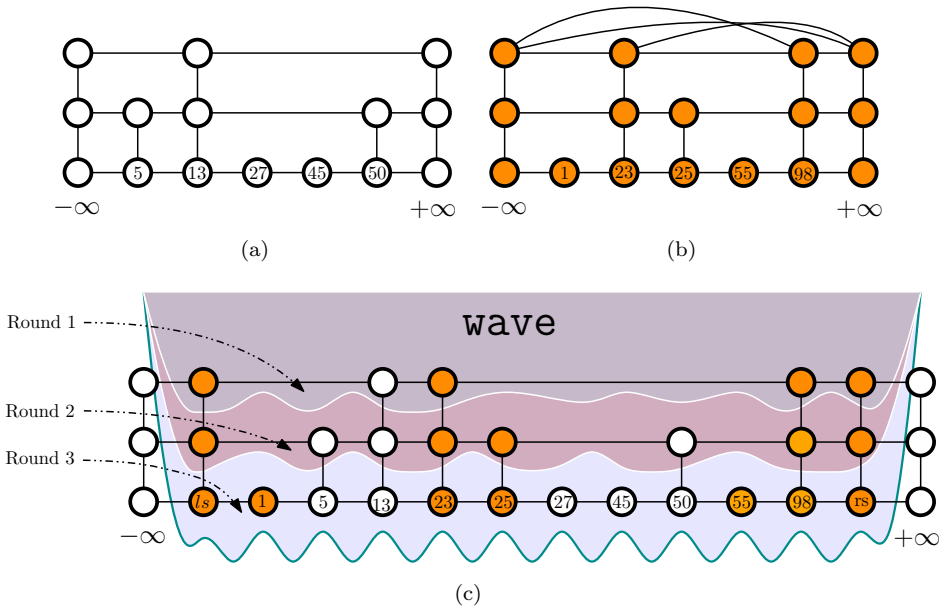


FIGURE 6.11: Example of the WAVE protocol. Figure 6.11(a) and Figure 6.11(b) show the clean network $\mathcal{C}$ (base skip list) and the preprocessed Buffer network $\mathcal{B}$, respectively. Figure 6.11(c) shows WAVE's execution: light gray, light brown, and light blue waves are the first, second and third rounds of the protocol's execution. After the last round, one step of rewiring is executed to remove the left and right sentries (ls, rs) from $\mathcal{C}$. In this example, the merging phase requires $\mathcal{O}(\log |\mathcal{B}|)$ rounds to merge the buffer network $\mathcal{B}$ with $\mathcal{C}$.

6.2.6 Update

After performing the previous phases, the live network $\mathcal{L}$ must be coupled with the newly updated clean network $\mathcal{C}$. A high level description on how to perform such update in constant time is the following: during the duplication phase a "snapshot" (e.g., a copy) of the clean network $\mathcal{C}$ is taken and used as the new $\mathcal{L}$. Intuitively, this procedure requires $\mathcal{O}(1)$ rounds because each node in $\mathcal{C}$ is taking care of the snapshot that can be considered as local computation. However, we must be careful with the notion of snapshot. To preserve dynamic resource-competitiveness we show how to avoid creating new edges while taking a snapshot of $\mathcal{C}$ during this phase. Assume that every edge in the clean network $\mathcal{C}$ has a length two local[11] binary label λ in which the first coordinate indicates its presence in $\mathcal{C}$ and the second one in $\mathcal{L}$. Moreover, each label λ can be of three different types: (1) "10" the edge is in $\mathcal{C}$ not in $\mathcal{L}$; (2) "01" the edge is not in $\mathcal{C}$ but it is in $\mathcal{L}$; and, (3) "11" the edge is in both $\mathcal{C}$ and $\mathcal{L}$. Moreover, assume that each node u in $\mathcal{C}$ has the label $\lambda(u, v)$ associated to its port encoding the connection with node v. Then, each node u in $\mathcal{C}$ for each $0 \leq \ell \leq \ell_{\max}(u)$ performs the following local computation:

[11]Meaning that each node $u \in \mathcal{C}$ has its own copy of the label λ for each edge incident to it. Observe that for an edge (u, v) the local label of u can not differ from the one of v.

1. If u's port-label associated to the edge connected to the left neighbor at level ℓ (i.e., $\lambda(u, v)$ such that $v \in N_{\mathcal{C}}^{(L,\ell)}(u)$) is "10" then set it to "11";

2. If u's port-label associated to the edge connected to the right neighbor at level ℓ (i.e., $\lambda(u, w)$ such that $w \in N_{\mathcal{C}}^{(R,\ell)}(u)$) is "10" then set it to "11";

Notice that during this phase no edge has the label "01" because $\mathcal{C}$ was "cleaned" during the deletion phase. All the nodes in $\mathcal{C}$ executed the labeling procedure, and the live network $\mathcal{L}$ is given by the edges with label "11". It follows that this phase requires a constant number of rounds.

Lemma 6.14. *The update phase requires constant number of rounds.*

Next, we show that all the phases in the maintenance cycle satisfy the dynamic resource competitiveness constraint defined in Section 6.1.3.

Lemma 6.15. *Each maintenance cycle is* (α, β)*-dynamic resource competitive with* $\alpha = \mathcal{O}(\log n)$ *and* $\beta = \mathcal{O}(polylog(n))$.

Proof. In order to prove that our maintenance protocol is dynamic resource competitive we suffice to show that a generic iteration of our maintenance cycle respects this invariant.

Let t_s be the time instant in which the deletion phase starts, and let $t_e = \mathcal{O}(t_s + \log n)$. The overall churn between the previous deletion phase and the current one is $C(t_s - \mathcal{O}(\log n), t_s) = \mathcal{O}(n)$, that is because we have $\mathcal{O}(n/\log n)$ churn at each round for $\mathcal{O}(\log n)$ rounds. The total amount messages sent during the phase is $\tilde{\mathcal{O}}(C(t_s - \mathcal{O}(\log n), t_s))$ and the number of formed edges is $\tilde{\mathcal{O}}(C(t_s - \mathcal{O}(\log n), t_s))$. Thus the overall amount of work during the deletion phase is $\mathcal{W}(t_s, t_e) = \tilde{\mathcal{O}}(C(t_s - \mathcal{O}(\log n), t_s))$.

During the buffer creation phase, we create a twin-butterfly network $\mathcal{M}$ in $\mathcal{O}(\log n)$ rounds using $\mathcal{O}(\log n)$ messages for each node at every round. Moreover, the twinbutterfly has $\mathcal{O}(n \log n)$ nodes of which $2n$ nodes have degree 4 while the rest have degree 8. This implies that while building the twinbutterfly we create roughly $\mathcal{O}(n \log n)$ edges and we exchange $\mathcal{O}(n \log^2 n)$ messages. Next, during each round of the AKS algorithm, each node u in $\mathcal{M}$ sends and receives a constant number of messages. Then, we copy such a list $\mathcal{O}(\log n)$ times w.h.p., in this way we create $\mathcal{O}(n \log n)$ edges. Finally, in the last step of the buffer creation phase, we run the deletion algorithm used in the previous phase on a subset of the nodes in the sorted list. Putting all together, during this phase, we have an overall amount of work of $\mathcal{W}(t_s, t_e) = \mathcal{O}(C(t_s - \mathcal{O}(\log n), t_s) \cdot \text{polylog}(n)) = \tilde{\mathcal{O}}(C(t_s - \mathcal{O}(\log n), t_s))$.

During the merge phase, we merge the buffer network with the clean one. The preprocessing step creates $\mathcal{O}(n \log n)$ edges on the buffer network using at most polylog(n) number of messages. During each step of the WAVE protocol sends $\mathcal{O}(n \cdot \text{polylog}(n))$ overall number of messages on the buffer network and creates $\mathcal{O}(n \log n)$ edges in the clean network. Putting all together, we have that the merge phase does not violate our dynamic work efficiency requirements. Indeed the overall work is bounded by $\mathcal{W}(t_s, t_e) = \mathcal{O}(C(t_s - \mathcal{O}(\log n), t_s) \cdot \text{polylog}(n)) = \tilde{\mathcal{O}}(C(t_s - \mathcal{O}(\log n), t_s))$.

Finally, during the duplication phase there is no exchange of messages and no edge is created.

We have that each phase of the maintenance cycle does not violate our resource competitiveness requirements in Definition 6.1. □

We conclude by showing that the skip list is maintained with high probability for n^c rounds, where $c \geq 1$ is an arbitrary big constant.

Lemma 6.16. *The maintenance protocol ensures that the resilient skip list is maintained effectively for at least poly(n) rounds with high probability.*

Proof. Each phase of the maintenance protocol (i.e., Algorithm 4) succeeds with probability at least $1 - \frac{1}{n^c}$, for some arbitrary big constant $c \geq 1$. Let X be a geometric random variable of parameter $p = \frac{1}{n^c}$ that counts the number of cycle needed to have the first failure, then its expected value is $\mathbb{E}[X] = n^c$, for $c \geq 1$. Moreover, the probability of maintenance protocol succeeding in a round n^r for some $r < c$ is $(1 - \frac{1}{n^c})^{n^r}$, Without loss of generality assume $c > 1$ and $r = c - 1$, then $(1 - \frac{1}{n^c})^{n^{c-1}} \leq e^{-n^{c-1}/n^c} \geq 1 - \frac{1}{n}$. □

Proof of the Main Theorem (Theorem 6.2). To prove the main theorem it is sufficient to notice that the skip list will always be connected and will never lose its structure thanks to Lemma 6.3. Indeed, every time a group of nodes is removed from the network, there are committees that in $\mathcal{O}(1)$ rounds will take over and act on their behalf. This allows all the queries to go through without any slowdown. Thus all the queries will be executed in $\mathcal{O}(\log n)$ rounds with high probability even with a churn rate of $\mathcal{O}(n/\log n)$ per round. Next we can show that the distributed data structure can be efficiently maintained using maintenance cycles of $\mathcal{O}(\log n)$ rounds each. To do this, it is sufficient to show that each phase of our maintenance cycle (Algorithm 4) can be carried out in $\mathcal{O}(\log n)$ rounds (see Lemma 6.4, Lemma 6.7, Lemma 6.13, and Lemma 6.14). Moreover, from Lemma 6.15 we have that each phase is (α, β)-dynamic resource competitive with $\alpha = \mathcal{O}(\log n)$ and $\beta = \mathcal{O}(\text{polylog}(n))$. Finally, the maintenance protocol

ensures that the distributed data structure is maintained for at least n^c rounds with high probability where $c \geq 1$ is an arbitrarily large constant.

6.2.7 Extending our approach to other data structures

The outlined maintenance protocol provides a convenient framework for constructing churn-resilient data structures. Furthermore, it can be easily adapted to maintain a more complex distributed pointer based data structure such as a skip graphs [7–10]. Indeed, with minor adjustments to the phases outlined in Algorithm 4, a maintenance cycle capable of maintain such data structures against *heavy churn rate* can be devised. To this end, we briefly discuss how to adapt our results to skip graphs. A skip graph [7], can be viewed as an extension of skip lists. Indeed, both consists of a set of increasingly sparse doubly-linked lists ordered by levels starting at level 0, where membership of a particular node u in a list at level ℓ is determined by the first ℓ bits of an infinite sequence of random bits associated with u, referred to as the *membership vector* of u, and denoted by $m(u)$. Let the first ℓ bits of $m(u)$ as $m(u)|\ell$. In the case of skip lists, level ℓ has only one list, for each ℓ, which contains all elements u such that $m(u)|\ell = 1^\ell$, i.e., all elements whose first ℓ coin flips all came up heads. Skip graphs, instead, have 2^ℓ lists at level ℓ, which we can index from 0 to $2^\ell - 1$. Node u belongs to the j-th list of level ℓ if and only if $m(u)|\ell$ corresponds to the binary representation of j. Hence, each node is present in one list of every level until it eventually becomes the only member of a singleton list. Without loss of generality, assume that the skip graph has sentry nodes as classical skip lists[12]. Thus, the skip graph can be maintained as follows:

Deletion. We can use the same $\mathcal{O}(\log n)$ rounds approach described in Section 6.2.3 to remove from the *clean skip graph* the nodes that have left the network.

Buffer Creation. To build the base level of the *buffer skip graph* we can use the same $\mathcal{O}(\log n)$ rounds approach described in Section 6.2.4. While, to construct the skip graph from level 0, we build an "augmented" skip list in which each node u in each level ℓ is identified by its ID/key and the list j at level ℓ in which u appears (there are 2^ℓ lists at level ℓ). Thus, each node u at some level ℓ can be: (i) *effective* in the list j and *fill-in* for the remaining $i \neq j$, for $j, i \in [1, 2^\ell]$; or, (ii) *fill-in* in all the lists $j \in [1, 2^\ell]$. Finally, such augmented skip list is transformed into a skip graph by running (in parallel) Algorithm 5 in Section 6.2.3 on each level ℓ for which there exists at least one fill-in node.

[12]Thus, at each level ℓ there are 2^ℓ sentinels (one for each list in such level).

Merge. We can use the same merge phase described in Section 6.2.5, with the only difference that a node u in the merge state at some level ℓ, must be merged in one of the 2^ℓ list in such level.

Duplicate. We can use the same approach described in Section 6.2.6.

Consequently, we can claim a similar result to Theorem 6.2 for skip graphs. In essence, with small changes in the delete, buffer creation, merge and update algorithms, our maintenance algorithm can be tailored to the specific data structure in question. Thus, making our approach ideal for building complex distributed data structures in highly dynamic networks.

6.2.8 Extending our approach to deal with multiple keys on each node

Throughout the paper we assumed that each node in the network posses *exactly* one element of the data structure. This assumption can be relaxed to deal with multiple keys on each node. Indeed, in the case in which in the network there are $t \cdot n$ keys, where $t > 0$, all the techniques described above allow to maintain the data structure in the presence of the same adversarial churn rate.

Corollary 6.17. *Given a network with n nodes in which each vertex v possesses t elements in the skip list for some $t > 1$. Then maintenance protocol requires $\mathcal{O}(t \cdot \log n)$ 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.*

Chapter 7

Conclusions

In this chapter, we summarize the results of this thesis and discuss future research directions.

In Chapter 3 we experimentally compared three global and three local proxies for shortest temporal betweenness rankings. One of these local proxies is a novel temporal degree notion, called the pass-through degree, which encodes the number of neighbor pairs that are temporally connected by a two-hop path passing through the node at hand. Our experimental results depict the performance of both global and local proxies in terms of running time and ranking quality. When applied to very large temporal networks, the pass-through degree clearly outperforms all the other competitors in terms of time performance. The pass-through degree achieves a time ratio that is around two, three, and four orders of magnitude better than Brandes, prefix-foremost temporal betweenness, and the fastest considered Onbra variant, respectively. In terms of ranking quality, the medians of the two time-intense Onbra variants are best, followed by Brandes, PTD, prefix-foremost temporal betweenness, and the fastest Onbra variant.

In Chapter 4 we proposed MANTRA, a novel framework for approximating the temporal betweenness centrality on large temporal networks. MANTRA relies on the state-of-the-art bounds on supremum deviation of functions based on the c-MCERA to provide a probabilistically guaranteed absolute ε-approximation of such centrality measure. Our framework includes a fast sampling algorithm to approximate the temporal diameter, average path length, and connectivity rate up to a small error with high probability. Such an approach is general and can be adapted to approximate several versions of these quantities based on different temporal path optimalities (e.g. [70, 71]). Our experimental results depict the performances of our framework versus the state-of-the-art algorithm for the temporal betweenness approximation. MANTRA consistently outperforms its competitor in terms of running time, sample size, and allocated memory.

Furthermore, our framework is the only available option to obtain meaningful temporal betweenness centrality approximations when we do not have access to servers with a large amount of memory. In the spirit of reproducibility, we developed an open-source framework in Julia that allows any user with an *average laptop* to approximate the temporal betweenness centrality on any kind of graph.

In Chapter 5 we introduced two models of dynamic random graphs inspired by the network formation protocol of the Bitcoin P2P network. We simulated the models to evaluate the structural properties of the snapshots of the dynamic graphs and to measure the time it takes a message starting at a random node to reach all, or almost all, the nodes. The results of our simulations show that the network structure generated by the E-RAES, by the V-RAES, and by a combination of the two models is globally very robust, in the sense that the network can quickly rebuild itself after node and edge failures. Moreover, the simulations of the flooding procedure show that the information spreading in the two models is fast and reliable, for the E-RAES essentially at any edge-failure rate, and for the V-RAES up to a node-failure rate as high as 70%. The outcomes of the simulations on the combined model EV-RAES are similar to those obtained in the V-RAES model for a large range of parameters. Since the degree of a full-node in the Bitcoin network is directly correlated to the amount of traffic going through the node, our results suggest that it is quite safe, for full-nodes of the Bitcoin network that need to reduce the bandwidth usage, to change the default value of the maximum number of connections from 125 to much smaller values. On the one hand, this significantly reduces the upload network traffic and, on the other hand, our simulations suggest that it does not compromise the overall stability and reliability of the network.

In Chapter 6 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-network 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 [7, 9, 10].

In a broader sense, our technique is general and can be used 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.

An additional contribution of our work is the improvement on the $\mathcal{O}(\log^3 n)$ rounds state-of-the-art technique for sorting in the Node-Capacitated Clique model [186]. We show how to sort n elements (despite a high churn rate) in $\mathcal{O}(\log n)$ rounds using results for sorting network theory. However, given the impracticability of the AKS sorting network, our result is purely theoretical. In practice, it could be easily implemented using Batcher's network instead of the AKS one. This change would slow down the bootstrap phase and the maintenance cycle to $\mathcal{O}(\log^2 n)$ rounds.

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.

There are many possible extensions of the contributions of this thesis and new directions for future research. We first discuss possible future investigations in the context of temporal graph mining.

A first future direction is explaining the correlations between our novel temporal degree notion (PTD) and the shortest temporal betweenness by using temporal graph parameters, such as the ones defined in the works of Tang et al. [187] and Nicosia et al. [22]. It would also be interesting to use PTD as a proxy for both static and temporal centralities in the context of routing schemes [188], as its local character enables an efficient distributed computation. From a theoretical point of view, possible directions of research include finding a conditional lower bound on the time complexity of computing the shortest temporal betweenness that is better than the lower bound implied by its non-temporal counterpart. Proving a conditional lower bound on the computation of the ego-network betweenness measures (or designing a better algorithm) is also a very challenging question. Moreover, the pass-through degree easily generalises to k-hop paths (instead of 2-hop paths). We believe that designing a quasi-linear time algorithm for computing such a generalisation, and verifying its quality in terms of proxying the shortest temporal betweenness, is the most natural continuation of this work. Furthermore, our novel algorithm MANTRA could be improved in terms of running time by combining our framework with the results by Zhang et al. [189] (WWW, 2024) and Brunelli et al. [190] (KDD, 2024) to speed up the temporal graph traversal for the shortest-temporal betweenness computation. In addition, a variant of MANTRA for the temporal edge betweenness could be employed to find communities in temporal graphs using a Girvan–Newman [191] approach to define a hierarchical clustering method to detect communities in temporal graphs. Such an algorithm would repeatedly alternate three phases: a phase of edge-betweenness estimation, a phase of temporal graph pruning in which k underlying edges with the highest temporal betweenness are

removed from the network, and a phase of temporal connectivity testing which will establish when to stop the "partitioning cycle". Moreover, MANTRA could potentially be used and extended to other temporal path-based centrality measures [37, 192]. Finally, our fast sampling-based approximation algorithm for the temporal distance-based metrics can be easily adapted to compute temporal-harmonic centrality in big temporal networks [192].

We continue the description of the possible extensions and contributions of this thesis by discussing possible future investigations in the context of efficient distributed algorithms in highly dynamic networks.

An interesting future direction for the work in Chapter 5, is to provide a rigorous theoretical analysis of some of the proposed dynamic graph models. A first step to provide such a formal analysis could be considering a different version of the Vertex-Dynamic RAES in which the churn is adversarial (see, for example [11]). We believe that for such a different setting, we can provide a theoretical analysis of the dynamic-graph model.

Another fundamental and extremely interesting future research direction is to maintain a distributed data structure in the presence of *Byzantine* nodes (i.e., malicious nodes) along with the churn. Considering both, Byzantine nodes and churn would make our result even more general and adaptable to a wide range of scenarios.

Appendix A

Appendix for Chapter 4

A.1 Extension of the $(\star)$-Temporal Betweenness Centrality to the edges.

We extend the concept of temporal betweenness to the temporal and static underlying edge of a given temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ as follows

Definition A.1 (Temporal Edge Betweenness)**.** The temporal betweenness of any edge $e = (u, v)$ in the *underlying graph* of a temporal graph $\mathcal{G}$ is defined as:

$$\mathrm{b}_e^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq z} \frac{\sigma_{sz}^{(\star)}(e)}{\sigma_{sz}^{(\star)}} \tag{A.1}$$

We define the temporal betweenness of a temporal edge as follows:

$$\mathrm{b}_{e,t}^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq z} \frac{\sigma_{sz}^{(\star)}(e,t)}{\sigma_{sz}^{(\star)}} \tag{A.2}$$

Where $\sigma_{sz}^{(\star)}(e,t)$ is the number of $(\star)$-temporal paths from s to z passing through the underlying edge[1] e at time t. Now we define the temporal edge betweenness of an underlying edge e as the sum of the temporal edge betweenness values of its appearances at time (e,t)

Lemma A.2. *For any underlying edge $e \in E$ it holds:*

$$b_e^{(\star)} = \frac{1}{n(n-1)} \sum_{t=0}^{T} b_{e,t}^{(\star)} \tag{A.3}$$

[1]Equivalently, let $e = (u, v)$, then $\sigma_{sz}^{(\star)}(e,t)$ is the number of $(\star)$-temporal paths from s to z passing through the temporal edge (u, v, t).

Proof.

$$\mathrm{b}_e^{(\star)} = \frac{1}{n(n-1)} \sum_{s \neq z} \frac{\sigma_{sz}^{(\star)}(e)}{\sigma_{sz}^{(\star)}} = \frac{1}{n(n-1)} \sum_{s \neq z} \sum_{t=0}^{T} \frac{\sigma_{sz}^{(\star)}(e,t)}{\sigma_{sz}^{(\star)}} = \frac{1}{n(n-1)} \sum_{t=0}^{T} \mathrm{b}_{e,t}^{(\star)}$$

□

Analogously to Lemma 4.2, we can show that the sum of the $(\star)$-temporal betweenness centrality of the underlying edges is equal to the average number of edges in a $(\star)$-temporal path. Given a temporal graph $\mathcal{G} = (V, \mathcal{E}, \mathcal{T})$ define

$$\Psi^{(\star)} = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{e \in E} \mathbb{1}[e \in tp_{sz}]$$

Where $\mathbb{1}[e \in tp_{sz}]$ assumes value 1 if and only if the underlying edge $e \in E$ appears in the temporal path tp_{sz}. Then the following lemma holds:

Lemma A.3. $\sum_{e \in E} b_e^{(\star)} = \Psi^{(\star)}$

Proof. Equation (A.1) can be rewritten as

$$\mathrm{b}_e^{(\star)} = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{tp \in \Gamma_{sz}^{(\star)}} \frac{\mathbb{1}[e \in \mathbf{Int}(tp)]}{\sigma_{sz}^{(\star)}}$$

Summing over the underlying edges $e \in E$ we obtain:

$$\sum_{v \in V} \mathrm{b}_e^{(\star)} = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{tp \in \Gamma_{sz}^{(\star)}} \frac{1}{\sigma_{sz}^{(\star)}} \sum_{e \in E} \mathbb{1}[e \in \mathbf{Int}(tp)]$$

$$= \frac{1}{n(n-1)} \sum_{s,z \in V} \frac{\sigma_{sz}^{(\star)}}{\sigma_{sz}^{(\star)}} \sum_{e \in E} \mathbb{1}[e \in \mathbf{Int}(tp_{sz})] = \frac{1}{n(n-1)} \sum_{s,z \in V} \sum_{e \in E} \mathbb{1}[e \in \mathbf{Int}(tp_{sz})] = \Psi^{(\star)}$$

□

Given Lemma A.2, Lemma A.3, and the temporal accumulation results in Buß et al. [3], we can adapt all the algorithms[2] for computing the $(\star)$-Temporal Vertex Betweenness centrality to compute the $(\star)$-Temporal Edge Betweenness of the underlying graph. Edge-temporal betweenness centrality can be used to develop fast temporal-community detection algorithms by using an approach similar to the well know Girwan-Newman algorithm [191]. Our approximation algorithms can be used to efficiently partition temporal graphs by removing the top-k underlying/temporal edges with highest temporal-betweenness scores and find communities. Moreover, given Lemma A.3, we have the

[2]Exact and approximate ones.

following corollary of Theorem 4.13 on the sufficient sample size for the $(\star)$-temporal edge betweenness:

Corollary A.4. *Let $\mathcal{F} = \{f(e), e \in E\}$ be a set of function from a domain $\mathcal{D}$ to $[0,1]$. Let $f(e)$ be a function such that $\mathbf{E}[f(e)] = b_e^{(\star)}$. Define $\hat{v} \in (0, 1/4]$ and $\Psi^{(\star)} \geq 0$ such that*

$$\max_{e \in E} Var(f(e)) \leq \hat{v} \qquad \text{and} \qquad \sum_{e \in E} b_e^{(\star)} \leq \Psi^{(\star)}$$

fix $\varepsilon, \delta \in (0,1)$, and let $\mathcal{S}$ be an i.i.d. sample taken from $\mathcal{D}$ of size

$$|\mathcal{S}| \in \mathcal{O}\left(\frac{\hat{v}+\varepsilon}{\varepsilon^2} \ln\left(\frac{\Psi^{(\star)}}{\delta \hat{v}}\right)\right)$$

It holds that $SD(\mathcal{F}, \mathcal{S}) \leq \varepsilon$ with probability $1-\delta$ over $\mathcal{S}$.

A.2 Other estimators for the $(\star)$-temporal betweenness centrality

In this section we present other two unbiased estimators for the $(\star)$-temporal betweenness centrality.

A.2.1 The Random Temporal Betweenness Estimator

We define the *Random Temporal Betweenness* estimator (**rtb**). An intuitive technique to obtain an approximation of the $(\star)$-temporal betweenness centrality of a temporal graph $\mathcal{G}$ is to run the exact temporal betweenness algorithm on a subset $\mathcal{S}$ of nodes selected uniformly at random from V. Thus, in this case, the sampling space $\mathcal{D}_{\mathbf{rtb}}$ is the set V of vertices in $\mathcal{G}$, and the distribution $\pi_{\mathbf{rtb}}$ is uniform over this set. The family $\mathcal{F}_{\mathbf{rtb}} = \{\widetilde{b}_{\mathbf{rtb}}^{(\star)}(v|s) : v \in V\}$, contains one function $\widetilde{b}_{\mathbf{rtb}}^{(\star)}(v|s)$ for each vertex v, defined as:

$$\widetilde{b}_{\mathbf{rtb}}^{(\star)}(v|s) = \frac{1}{n-1} \cdot \sum_{\substack{z \in V \\ z \neq s}} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}} \in [0,1] \tag{A.4}$$

It follows that

Lemma A.5. *The **rtb** is an unbiased estimator of the $(\star)$-temporal betweenness centrality.*

Proof.

$$\mathbf{E}\left[\widetilde{b}^{(\star)}_{\mathbf{rtb}}(v|s)\right] = \sum_{s\in V} \mathbf{Pr}\,(s)\cdot \widetilde{b}^{(\star)}_{\mathbf{rtb}}(v|s) = \sum_{s\in V}\frac{1}{n}\left(\frac{1}{n-1}\sum_{z\neq s\neq v}\frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}}\right)$$

□

The function $\widetilde{b}^{(\star)}_{\mathbf{rtb}}(v|s)$ is computed by performing a full $(\star)$-temporal breadth first search visit from s, and then backtracking along the temporal directed acyclic graph as in the exacts algorithms [3]. The **rtb** framework computes *all* the sets $\Gamma^{(\star)}_{sz}$ from the sampled vertex s to *all* other vertices $z \in V$ using a full $(\star)$-TBFS. Moreover, in a worst-case scenario this algorithm could touch all the temporal edges in the temporal graph at *every sample* making the estimation process slow. As for the static case [193, 194], this algorithm does not scale well as the temporal network size increases.

A.2.2 The Temporal Riondato and Kornaropoulos estimator

We extend the estimator for static graphs by Riondato and Kornaropoulos in [195] to the temporal setting. The algorithm, (1) computes the set $\Gamma^{(\star)}_{sz}$ as **ob**; (2) randomly selects a $(\star)$-temporal path tp_{sz} from $\Gamma^{(\star)}_{sz}$; and, (3) increases by $\frac{1}{r}$ the temporal betweenness of each vertex v in $\mathbf{Int}(\mathrm{tp})$ (where r is the sample size). The procedure to select a random temporal path from $\Gamma^{(\star)}_{sz}$ is inspired by the dependencies accumulation to compute the *exact* temporal betweenness scores by Buß et al.[3]. Let s and z be the vertices sampled by our algorithm. We assume that s and z are temporally connected, otherwise the only option is to select the empty temporal path $tp_{\emptyset}$. Given the set $\Gamma^{(\star)}_{sz}$ of all the $(\star)$-temporal paths from s to z, first we notice that the truncated $(\star)$-TBFS from s to z produces a time respecting tree from the vertex appearance $(s,0)$ to all the vertex appearances of the type (z,t_z) for some t_z. Let tp^* be the sampled $(\star)$-temporal path we build *backwards* starting from one of the temporal endpoints of the type (z,t_z) for some t_z. First, we sample such (z,t_z) as follows: a vertex appearance (z,t_z) is sampled with probability $\sigma^{t_z}_{sz}/(\sum_t \sigma^t_{sz}) = \sigma^{t_z}_{sz}/\sigma_{sz}$, where σ^t_{sz} is the number of $(\star)$-temporal paths reaching z from s at time t. Assume that (z,t_z) was put in the sampled path tp^*, i.e., $tp^* = \{(z,t_z)\}$. Now we proceed by sampling one of the temporal predecessors (w,t_w) in the *temporal predecessors* set $P(z,t_z)$ with probability $\sigma^{t_w}_{sw}/(\sum_{(x,t)\in P(z,t_z)}\sigma^t_{sx})$. After putting the sampled vertex appearance, let us assume (w,t_w), in tp^* we iterate the same process through the predecessors of (w,t_w) until we reach $(s,0)$.

Theorem A.6. *Let $tp^*_{sz} \in \Gamma^{(\star)}_{sz}$ be the $(\star)$-temporal path sampled using the above procedure. Then, the probability of sampling tp^*_{sz} is $\mathbf{Pr}\,(tp^*) = \frac{1}{\sigma^{(\star)}_{sz}}$*

Proof. Let us simplify the notation with $\sigma_{sz} = \sigma_{sz}^{(\star)}$. The probability of getting such tp^* using the aforementioned temporal path sampling technique is:

$$\mathbf{Pr}\left(tp^*\right) = \frac{\sigma_{sz}^{t_z}}{\sum_t \sigma_{sz}^t} \cdot \frac{\sigma_{sw_{k-1}}^{t_{w_{k-1}}}}{\sum_{(x,t)\in P(z,t_z)} \sigma_{sx}^t} \cdot \frac{\sigma_{sw_{k-2}}^{t_{w_{k-2}}}}{\sum_{(x,t)\in P(w_{k-1},t_{k-1})} \sigma_{sx}^t} \cdots$$
$$\frac{\sigma_{sw_1}^{t_{w_1}}}{\sum_{(x,t)\in P(w_2,t_2)} \sigma_{sx}^t} \cdot \frac{1}{\sum_{(x,t)\in P(w_1,t_1)} \sigma_{sx}^t}$$

Observe that $\sigma_{sw}^{t_w} = \sum_{(x,t)\in P(w,t_w)} \sigma_{sx}^t$ and that $\sum_t \sigma_{sz}^t = \sigma_{sz}$. Thus, the formula can be rewritten as follows:

$$\mathbf{Pr}\left(tp^*\right) = \frac{\sigma_{sz}^{t_z}}{\sigma_{sz}} \cdot \frac{\sigma_{sw_{k-1}}^{t_{w_{k-1}}}}{\sigma_{sz}^{t_z}} \cdots \frac{1}{\sigma_{sw_1}^{t_{w_1}}} = \frac{1}{\sigma_{sz}}$$

and the fact that (if the temporal graph has no self loop) for (w_1, t_{w_1}), which is a temporal neighbor of $(s, 0)$, $\sigma_{sw_1} = 1$. □

Observe that each $tp_{sz} \in \mathbb{TP}_{\mathcal{G}}^{(\star)}$ is sampled according to the function $\pi_{\mathbf{trk}}(tp_{sz}) = \frac{1}{n(n-1)} \frac{1}{\sigma_{sz}^{(\star)}}$ which (according to Theorem A.7) is a valid probability distribution over $\mathcal{D}_{\mathbf{trk}} = \mathbb{TP}_{\mathcal{G}}^{(\star)}$.

Theorem A.7. *The function $\pi_{\boldsymbol{trk}}(tp_{sz})$, for each $tp_{sz} \in \mathcal{D}_{\boldsymbol{trk}}$, is a valid probability distribution.*

Proof. Let $\Gamma_{sz}^{(\star)}$ be the set of $(\star)$-optimal temporal paths from s to z where $s \neq z$. Then,

$$\sum_{tp_{sz}\in\mathcal{D}_{\mathbf{trk}}} \pi(tp_{sz}) = \sum_{tp_{sz}\in\mathcal{D}_{\mathbf{trk}}} \frac{1}{n(n-1)} \frac{1}{\sigma_{sz}^{(\star)}} = \sum_{s\in V} \sum_{\substack{z\in V \\ s\neq z}} \sum_{tp_{sz}\in\Gamma_{sz}^{(\star)}} \frac{1}{n(n-1)} \frac{1}{\sigma_{sz}^{(\star)}}$$
$$= \sum_{s\in V} \sum_{\substack{z\in V \\ s\neq z}} \frac{1}{n(n-1)} \frac{\sigma_{sz}^{(\star)}}{\sigma_{sz}^{(\star)}} = \frac{1}{n(n-1)} \sum_{s\in V} \sum_{\substack{z\in V \\ s\neq z}} 1 = \frac{1}{n(n-1)} \sum_{s\in V} (n-1) = 1$$

□

For $tp_{sz} \in \mathcal{D}_{\mathbf{trk}}$, and for all $v \in V$ define the family of functions $\mathcal{F}_{\mathbf{trk}} = \{f_{\mathbf{trk}}^{(\star)}(v) : v \in V\}$ where $\widetilde{b}_{\mathbf{trk}}^{(\star)}(v|tp_{sz}) = \mathbb{1}\left[v \in \ \mathbf{Int}(tp_{sz})\right]$. Observe that

Lemma A.8. *For $\widetilde{b}_{\boldsymbol{trk}}^{(\star)}(v) \in \mathcal{F}$ and for all $tp_{sz} \in \mathcal{D}_{\boldsymbol{trk}}$, such that each tp_{sz} is sampled according to the probability function $\pi(tp_{sz})$, then $\mathbf{E}\left[\widetilde{b}_{\boldsymbol{trk}}^{(\star)}(v|tp_{sz})\right] = b_v^{(\star)}$.*

Proof.

$$\mathbf{E}_{tp_{sz}\in\mathcal{D}_{\mathbf{trk}}}\left[\widetilde{b}^{(\star)}_{\mathbf{trk}}(v|tp_{sz})\right] = \sum_{tp_{sz}\in\mathcal{D}_{\mathbf{trk}}} \pi(tp_{sz})\widetilde{b}^{(\star)}_{\mathbf{trk}}(v|tp_{sz}) = \sum_{tp_{sz}\in\mathcal{D}_{\mathbf{trk}}} \frac{\widetilde{b}^{(\star)}_{\mathbf{trk}}(v|tp_{sz})}{n(n-1)\sigma^{(\star)}_{sz}}$$
$$= \frac{1}{n(n-1)} \sum_{\substack{s,z\in V\\ s\neq v\neq z}} \sum_{tp\in\Gamma^{(\star)}_{sz}} \frac{\mathbb{1}\left[v \in \mathbf{Int}(tp)\right]}{\sigma^{(\star)}_{sz}} = \frac{1}{n(n-1)} \sum_{\substack{s,z\in V\\ s\neq v\neq z}} \frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}}$$

□

A.2.3 Statistical Properties of the estimators

In this section, we discuss the differences between **ob**, **trk**, and **rtb** estimators in terms of statistical properties. Furthermore, since they share *the same* properties with their static counterparts, all the results in [77] directly extend to the aforementioned estimators. The key observation is that the three estimators, being unbiased, are equal to the the exact $(\star)$-temporal betweenness in expectation, and each simply commutes gradually less randomness from the inner sample to the outer expectations. Thus, running MANTRA with these different estimators may be seen as progressively computing more random stochastic approximation of the exact algorithm. To see why this is true, we observe that, by running MANTRA's estimation phase with $r = 1$ samples, the relationship between these estimators can be described as follows:

$$b^{(\star)}_v = \underbrace{\mathbf{E}_{s\in V}\left[\overbrace{\mathbf{E}_{s\neq z}\left[\underbrace{\mathbf{E}_{tp_{sz}\in\Gamma^{(\star)}_{sz}}\left[\widetilde{b}^{(\star)}_{\mathbf{trk}}(v|tp_{sz})\right]}_{\mathbf{trk}}\right]}^{\mathbf{ob}}\right]}_{\mathbf{rtb}}$$

where s is sampled uniformly at random from V, z form $V \setminus \{s\}$, and tp_{sz} is sampled uniformly at random from $\Gamma^{(\star)}_{sz}$ if s and z are temporally connected, otherwise $tp_{sz} = \{\emptyset\}$. This suggests that each estimator computes a conditional expectation as a proxy for the (unconditional) expectation of the temporal betweenness centrality.

A well established way to compare unbiased estimators for the same quantity is to compare their variances[3]. The results by Cousins et al., [77] directly apply to the temporal setting. We have the following corollary:

[3]This also gives information about the (expected) Mean Squared Error of the estimators.

Corollary A.9. *For any $v \in V$ it holds*

$$\mathbf{Var}\left[\frac{\tilde{b}^{(\star)}_{\boldsymbol{ob}}(v)}{\tilde{b}_{\boldsymbol{trk}}(v)^{(\star)}}\right] \leq \max_{\substack{s \\ s\neq z}} \frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}} \in [0,1]$$

$$\mathbf{Var}\left[\frac{\tilde{b}^{(\star)}_{\boldsymbol{rtb}}(v)}{\tilde{b}_{\boldsymbol{trk}}(v)^{(\star)}}\right] \leq \max_{s} \mathbf{E}_{z\neq v}\left[\frac{\sigma^{(\star)}_{sz}(v)}{\sigma^{(\star)}_{sz}}\right] \in [0,1]$$

and,

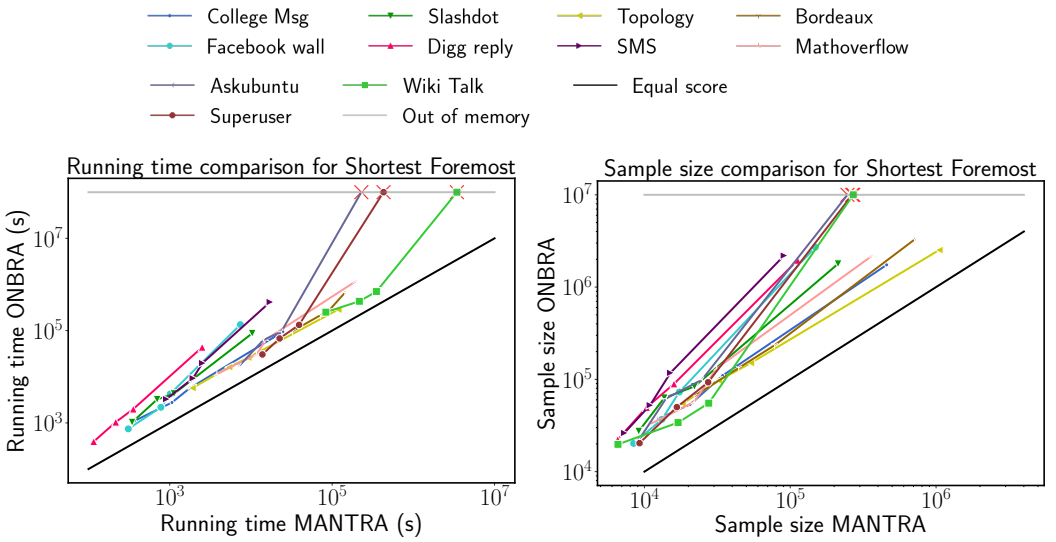
$$\mathbf{Var}\left[\frac{\tilde{b}^{(\star)}_{\boldsymbol{rtb}}(v)}{\tilde{b}^{(\star)}_{\boldsymbol{ob}}(v)}\right] \leq 1$$

The corollary tells us that **rtb** is the estimator that has lowest variance among all of them. This is not surprising because **rtb** collects more information per sample compared to the remaining ones. However, this strength is also its major weakness because the **rtb** requires *full* $(\star)$-temporal BFSs that are much slower than the truncated ones used by **ob** and **trk**. Between, these last two estimators, **ob** is the one that collects more information. That is because, once the truncated temporal BFS from s to z is completed, **ob** updates the temporal betweenness of every encountered node $v \neq s \neq z$.

A.3 Additional Experiments

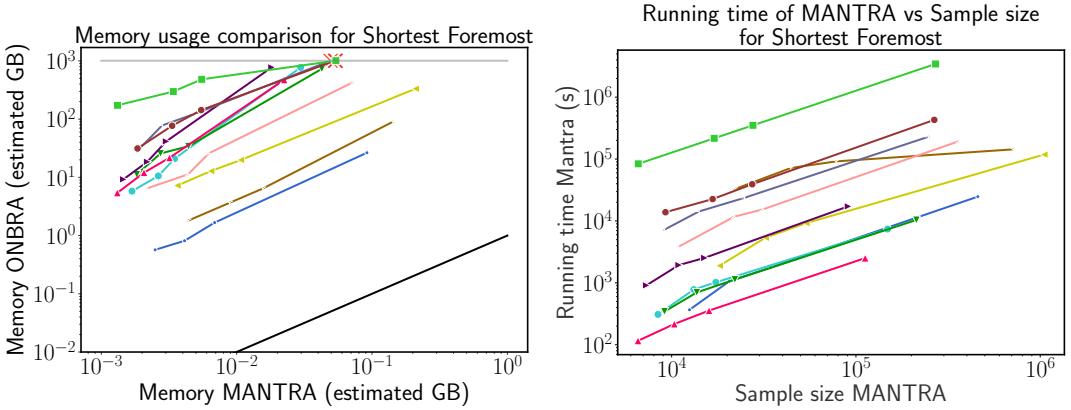
A.3.1 Experiments for the shortest (foremost)-temporal betweenness

Here we provide the results of the experiments in Section 4.4 for the shortest-foremost temporal betweenness centrality. As previously mentioned, these plots follow the same trend of the ones for the prefix-foremost and shortest temporal betweenness. Moreover, we display the experiments for small values of ε that have been excluded from Chapter 4.



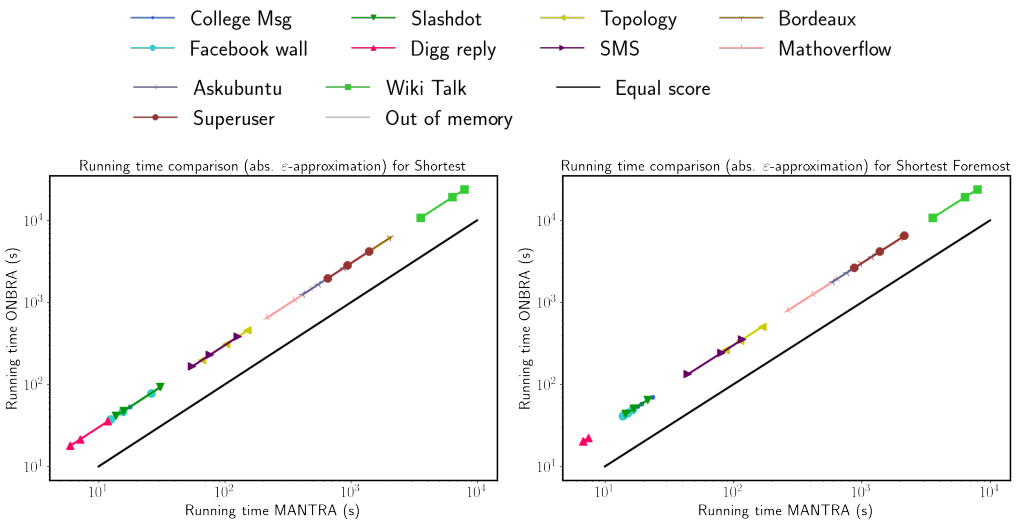
(a) (b)

FIGURE A.1: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$ For the Shortest-foremost temporal betweenness. Comparison between the running times and sample sizes of ONBRA and MANTRA **(a-b)**.



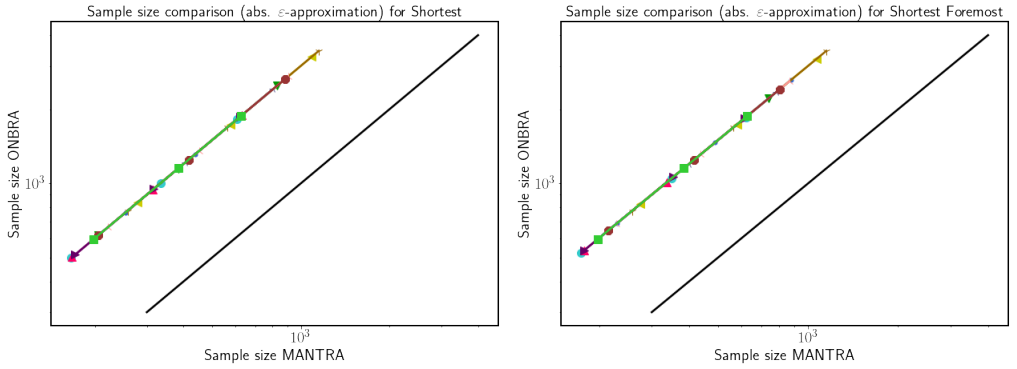
(a) (b)

FIGURE A.2: Experimental analysis for $\varepsilon \in \{0.01, 0.007, 0.005, 0.001\}$ For the Shortest-foremost temporal betweenness. Comparison between the allocated memory **(a)** of ONBRA and MANTRA, and the relation between running time and sample size for MANTRA **(b)**.



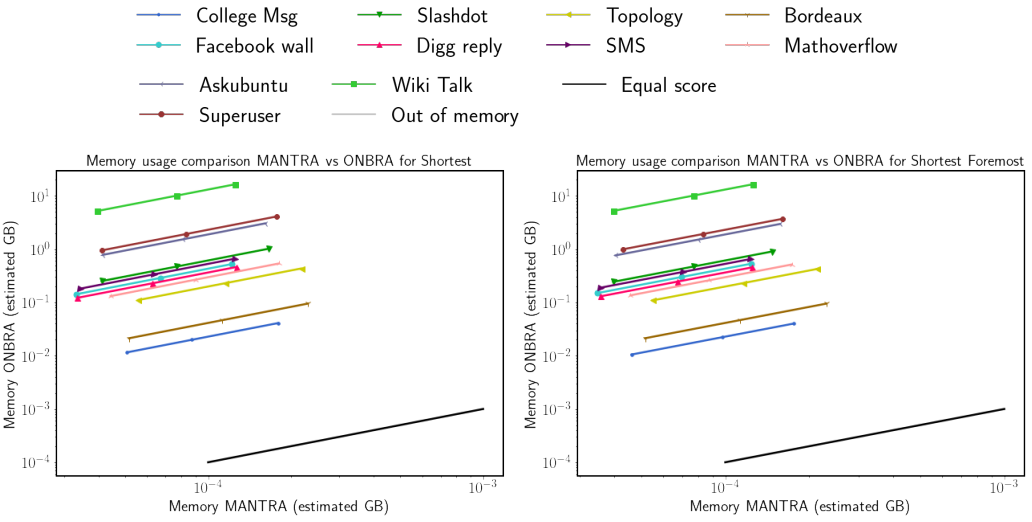
(a) (b)

FIGURE A.3: Experimental analysis for $\varepsilon \in \{0.1, 0.07, 0.05\}$ For the Shortest and Shortest-foremost temporal betweenness. Comparison between the running times **(a-b)** of ONBRA and MANTRA.



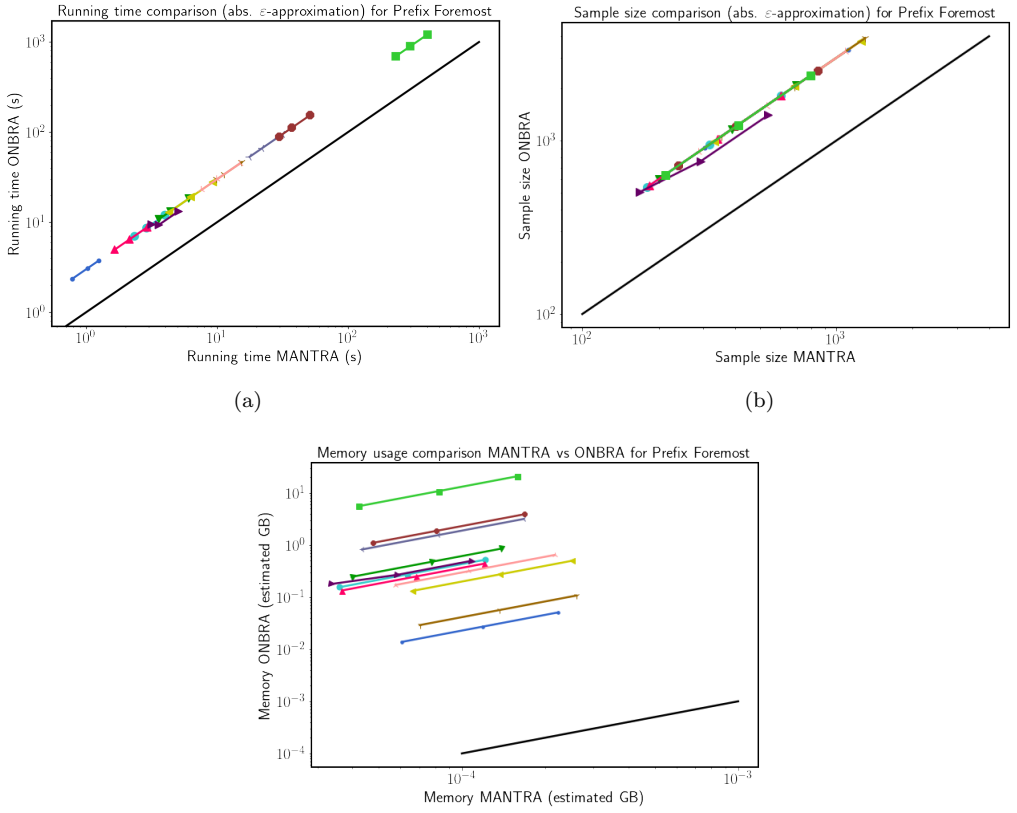
(a) (b)

FIGURE A.4: Experimental analysis for $\varepsilon \in \{0.1, 0.07, 0.05\}$ For the Shortest and Shortest-foremost temporal betweenness. Comparison between the sample sizes **(a-b)** of ONBRA and MANTRA.



(a) (b)

FIGURE A.5: Experimental analysis for $\varepsilon \in \{0.1, 0.07, 0.05\}$ For the Shortest and Shortest-foremost temporal betweenness. Comparison between the allocated space **(a-b)** of ONBRA and MANTRA.



(a) (b)

(c)

FIGURE A.6: Experimental analysis for $\varepsilon \in \{0.1, 0.07, 0.05\}$ For the Prefix-foremost temporal betweenness. Comparison between the running times **(a)**, sample sizes **(b)**, and allocated memory **(c)** of ONBRA and MANTRA.

Appendix B

Appendix for Chapter 5

B.1 Experiments for other values of d and c

In this section, we provide the results obtained by simulating E/V/EV-RAES with other values of d and c. More precisely, we show the behavior of the models with $d = 3, c = 3$, and with an "interpolated" setting between $d = 4, c = 1.5$ and the default parameters of the Bitcoin Protocol ($d = 8, c = 15.625$). These experiments show that the results are qualitatively very similar to the ones for $d = 4$ and $c = 1.5$.

B.1.1 $d = 3$ and $c = 3$

In this section, we present the results of the simulations for all the models using $d = 3$, and $c = 3$. Moreover, it can be noticed that the plots have a similar trend of the ones for $d = 4$, and $c = 1.5$. Figures B.1, B.2 show, respectively, the same experiments performed for the E-RAES model in Section 5.3.2 and Section 5.3.3. While, Figures B.3,B.4 show the results of the simulations for the V-RAES in Section 5.4.1, and 5.4.2

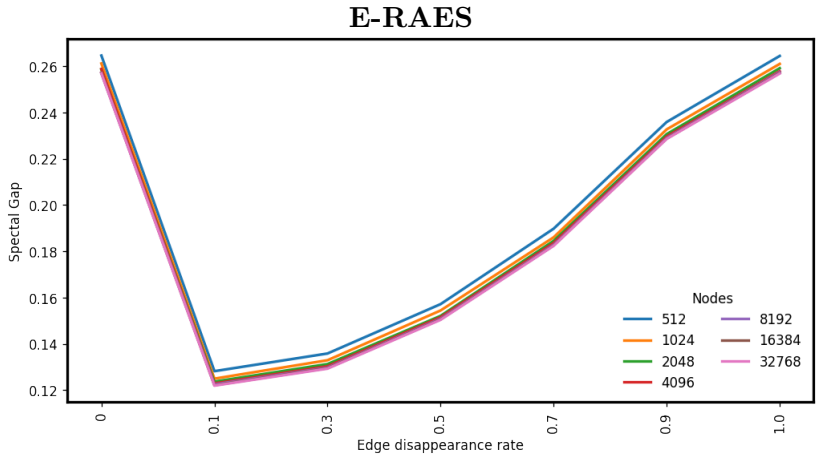


FIGURE B.1: Average spectral gap for E-RAES of 10 runs of 100 rounds each, for $d = 3, c = 3$, and increasing values for number of nodes n and edge-failure probability p. The spectral gap is computed before the edge failure step.

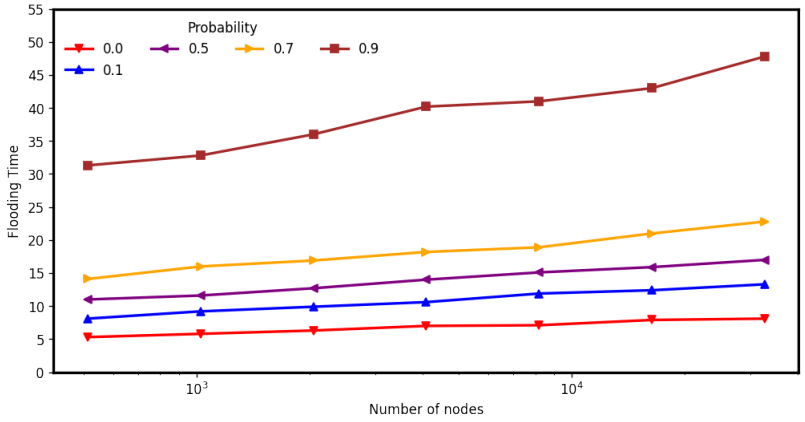


FIGURE B.2: Semi-log plot of the average flooding time (over 10 runs) with $2^9 \leq n \leq 2^{15}, p \leq 0.9$.

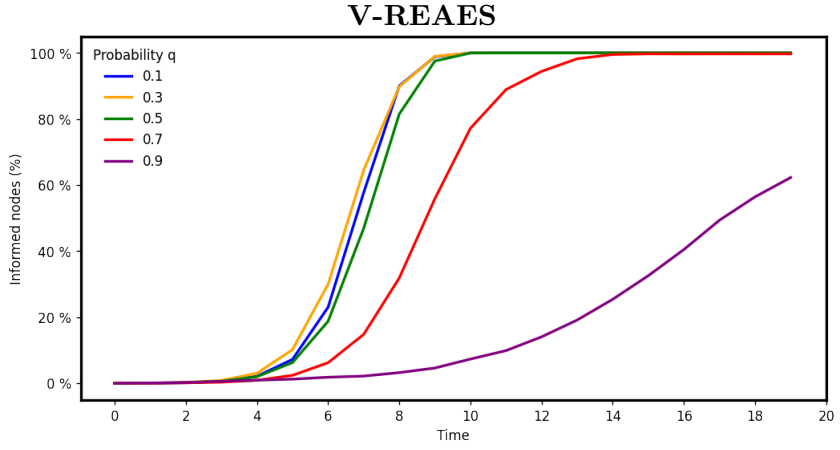


FIGURE B.3: Average over 10 runs of the evolution of the fraction of informed nodes α_t, at each time step. The ratio λ/q is fixed to 2^{15}.

Figures B.5,B.6, show, the results of the EV-RAES simulations using $d = 3$, $c = 3$ and the same values of Section 5.5 for λ/q, p, and q.

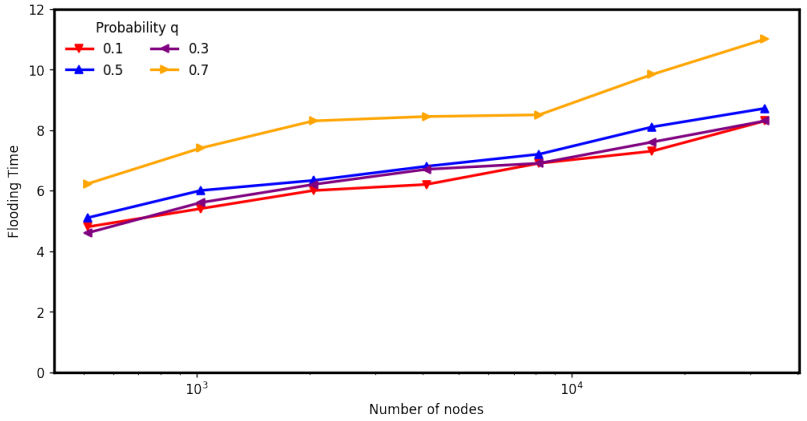


FIGURE B.4: Semi-log-plot of the average flooding time trend (over 10 runs) of the V-RAES with $2^9 \leq \lambda/q \leq 2^{15}$.

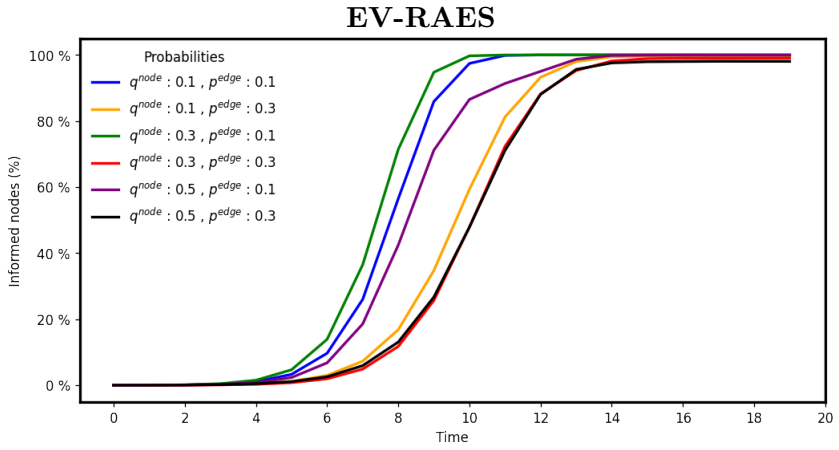


FIGURE B.5: Evolution of the fraction of informed nodes α_t. The ratio λ/q is fixed to 2^{15}.

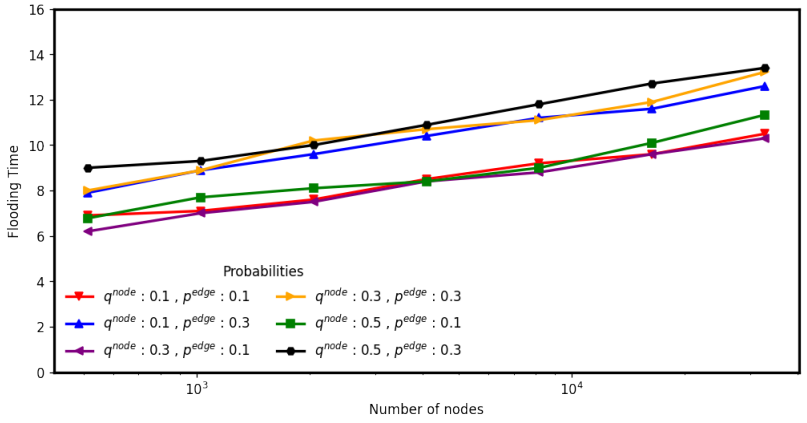


FIGURE B.6: Semi-log-plot of the average flooding time (over 10 runs) of the EV-RAES with $2^9 \leq \lambda/q \leq 2^{15}$, node disappearance rate $q = 0.1, 0.3, 0.5$, and edge disappearance rate $p = 0.1, 0.3$.

B.1.2 $d = 6$ and $c = 5$

In this section, we present the results of the simulations for all the models using $d = 6$, and $c = 5$. Moreover, it can be noticed that the plots have a similar trend of the ones for $d = 4$, and $c = 1.5$. Figures B.7, B.8 show, respectively, the same experiments performed for the E-RAES model in Section 5.3.2 and Section 5.3.3. While, Figures B.9,B.10 show the results of the simulations for the V-RAES in Section 5.4.1, and 5.4.2

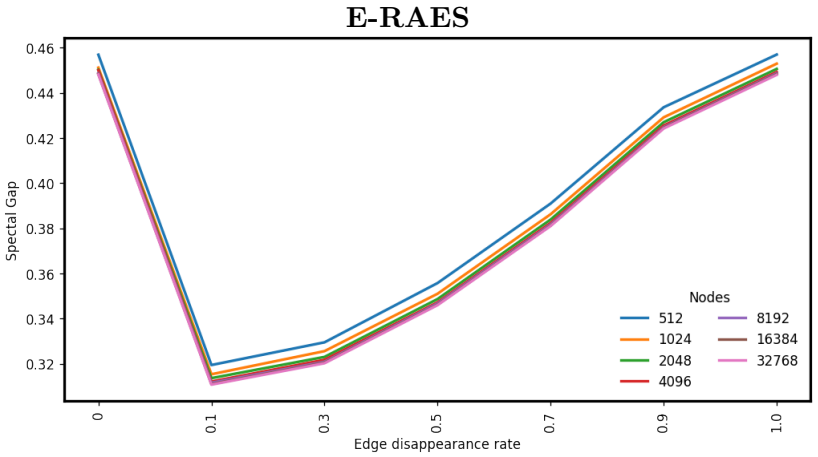


FIGURE B.7: Average spectral gap for E-RAES of 10 runs of 100 rounds each, for $d = 6, c = 5$, and increasing values for number of nodes n and edge-failure probability p. The spectral gap is computed before the edge failure step.

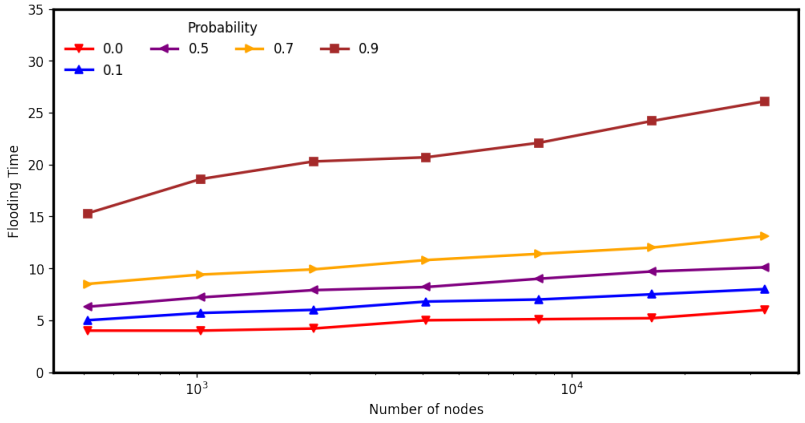


FIGURE B.8: Semi-log plot of the average flooding time (over 10 runs) with $2^9 \leq n \leq 2^{15}, p \leq 0.9$.

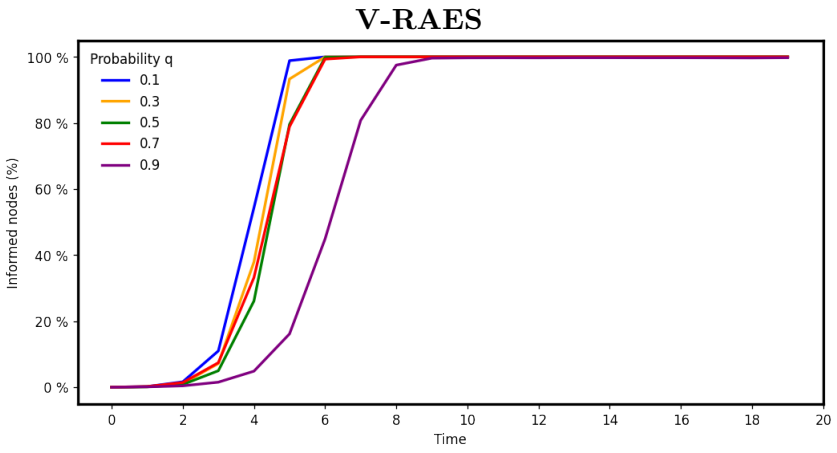


FIGURE B.9: Average over 10 runs of the evolution of the fraction of informed nodes α_t, at each time step. In the plots the ratio λ/q is fixed to 2^{15}.

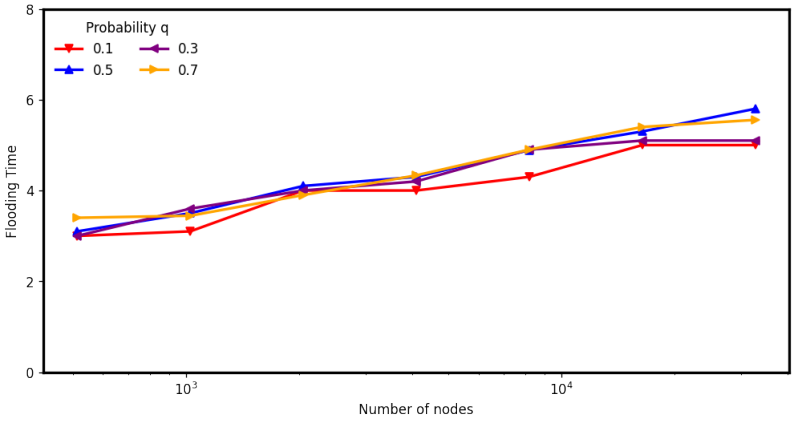


FIGURE B.10: Semi-log-plot of the average flooding time trend (over 10 runs) of the V-RAES with $2^9 \leq \lambda/q \leq 2^{15}$.

Figures B.11,B.12, show, the results of the EV-RAES simulations using $d = 6$, $c = 5$ and the same other values for $\lambda/q, q$, and p of Section 5.5.

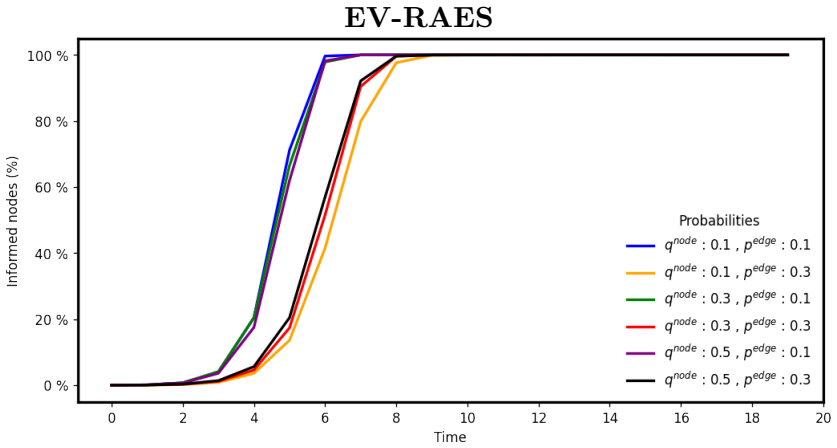


FIGURE B.11: Evolution of the fraction of informed nodes α_t. The ratio λ/q is fixed to 2^{15}.

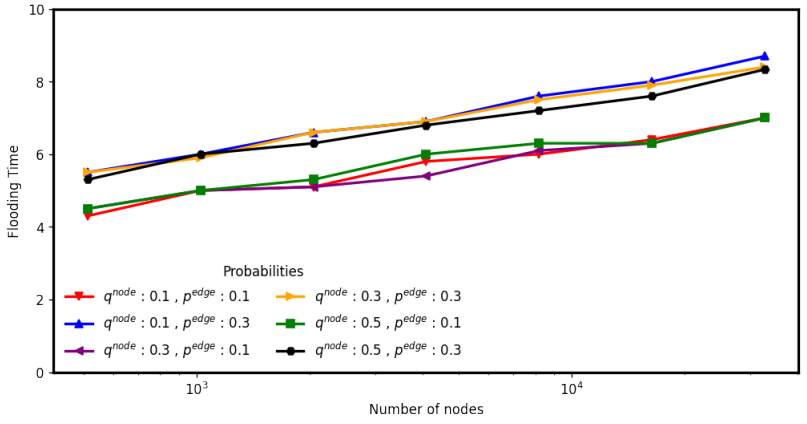


FIGURE B.12: Semi-log-plot of the average flooding time (over 10 runs) of the EV-RAES with $2^9 \leq \lambda/q \leq 2^{15}$, node disappearance rate $q = 0.1, 0.3, 0.5$, and edge disappearance rate $p = 0.1, 0.3$.

Appendix C

Appendix for Chapter 6

C.1 Useful Skip list properties

Lemma C.1. *The height of a n-element skip list is $\mathcal{O}(\log n)$ w.h.p.*

Proof. Let Y_i for $i \in [n]$ be the random variable that counts the number of consecutive heads we obtain while tossing a p-biased coin before we get a tail. Moreover, define h to be the maximum height of the skip list i.e., $h = 1 + \max\{Y_i : i \in [n]\}$. Observe that $\mathbf{Pr}(Y_i \geq k) = p^{k-1}$, and that by a straightforward application of the union bound we obtain the probability of having a skip list of height at least k, $\mathbf{Pr}(h \geq k) \leq np^{k-1}$. Choosing $k = 4\log_{1/p} n + 1$, we obtain a high confidence bound for the number of levels in the distributed data structure

$$\mathbf{Pr}\left(h \geq 4\log_{1/p} n + 1\right) \leq n(1/n)^4 = 1/n^3 \tag{C.1}$$

Moreover, h's expected values is

$$\mathbb{E}[h] = \sum_{i>0} \mathbf{Pr}(h \geq i) = \sum_{i=1}^{4\log n} \mathbf{Pr}(h \geq i) + \sum_{i>4\log n} \mathbf{Pr}(h \geq i) \leq \sum_{i=1}^{4\log n} 1 + \sum_{i>4\log n} np^{i-1} \tag{C.2}$$

$$\leq 4\log n + np^{4\log n}\left(\sum_{i\geq 1} p^{i-1}\right) = 4\log n + n\left(\frac{1}{n^4}\right)\left(\frac{1}{1-p}\right) \leq 4\log n + 1 = \mathcal{O}(\log n) \tag{C.3}$$

Where the last inequality holds if the log is in base $1/p$ and if n is sufficiently large. □

Next, we say that $X_\ell = \{x_1, x_2, \ldots, x_k\}$ is a *run*/cohesive group of nodes in a skip list $\mathcal{L}$ at level ℓ, if X_ℓ is a set of consecutive nodes such that $\ell^{\mathcal{L}}_{\max}(x_i) = \ell$ for each $x_i \in X_\ell$. Moreover, we give a high confidence bound on the size of a run X_ℓ.

Lemma C.2. *The size of a run of nodes X_ℓ for some level ℓ, is at most $\mathcal{O}(\log n)$ w.h.p., and its expected value is $1/(1-p)$.*

Proof. The size of a run of nodes $|X_\ell|$ is a geometric random variable with parameter $1-p$. Thus,

$$\mathbf{Pr}\left(|X_\ell| \geq k\right) \leq (1-p)^{k-1} \tag{C.4}$$

Choosing $k = c\log_{1-p} n + 1$ for $c \geq 1$, gives us a high confidence bound on the number of consecutive nodes in a run of node X_ℓ at some level ℓ. Indeed,

$$\mathbf{Pr}\left(|X_\ell| \geq c\log_{1-p} n + 1\right) \leq 1/n^c \tag{C.5}$$

Moreover, its expected value is $\mathbb{E}\left[|X_\ell|\right] = 1/(1-p)$, assuming $p = 1/2$ (in other words, constant), we have that the expected length of a run of nodes at some level ℓ of the skip list is $\mathcal{O}(1)$. □

Finally, we conclude with the analysis of the search/deletion/insertion of an element in a skip list (see Figure C.1 for an example about one of these operations).

Lemma C.3. *The running time for an insertion, deletion and search of an element in a skip list takes $\mathcal{O}(\log n)$ rounds w.h.p.*

Proof. Let R_i be number of horizontal edges at level $0 \leq i \leq h$ crossed by a search operation that starts on the left topmost node of the skip list. Define the random variable $W_h = |R_0| + |R_1| + \cdots + |R_h|$ to be the amount of horizontal moves performed by the search algorithm. Observe that each R_i is a geometric random variable of parameter $1-p$ and that h is a random variable itself. From Lemma C.1 we know that $\mathbb{E}\left[h\right] = \mathcal{O}(\log n)$ and that $\mathbf{Pr}\left(h \geq 4\log n + 1\right) \leq \frac{1}{n^3}$. Since h is a random variable, W_h is a *random sum* of random variables thus we can not make a straightforward use of the properties of sum

of geometric random variables, rather we can write:

$$\mathbf{Pr}\left(W_h > 16\log n\right) = \mathbf{Pr}\left(W_h > 16\log n \cap h \leq 4\log n\right) + \mathbf{Pr}\left(W_h > 16\log n \cap h > 4\log n\right) \tag{C.6}$$

$$\leq \sum_{h=0}^{4\log n} \mathbf{Pr}\left(W_h > 16\log n\right) + \mathbf{Pr}\left(h > 4\log n\right) \leq (1 + 4\log n)\mathbf{Pr}\left(W_{4\log n} > 16\log n\right) + \frac{1}{n^3} \tag{C.7}$$

We notice that $W_{4\log n}$ is a deterministic sum of geometric random variables. Hence we can use the relation between the upper tail value of a negative binomial distribution and the lower tail value of a suitably defined binomial distribution to derive upper tail estimates for the negative binomial distribution (see Appendix **??**). Thus, we rewrite $\mathbf{Pr}\left(W_{4\log n} > 16\log n\right)$ as $\mathbf{Pr}\left(Y \leq 4\log n\right)$ where Y is a binomial random variable with parameter $n = 16\log n$ and p. Now, we apply a Chernoff bound [39] and by setting $k = 4\log n$, $p = 1/2$ and $\mu = 8\log n$ we obtain

$$\mathbf{Pr}\left(W_{4\log n} > 16\log n\right) = \mathbf{Pr}\left(Y \leq 4\log n\right) \leq \left(\frac{8\log n}{4\log n}\right)^{4\log n}\left(\frac{8\log n}{12\log n}\right)^{12\log n} = \frac{2^{16\log n}}{3^{12\log n}} \tag{C.8}$$

$$= \left(\frac{16}{27}\right)^{4\log n} \leq \frac{1}{n^3} \tag{C.9}$$

Therefore,

$$\mathbf{Pr}\left(W_h > 16\log n\right) \leq (1 + 4\log n)\left(\frac{1}{n^3}\right) + \frac{1}{n^3} < \frac{1}{n^2} \quad \text{if } n > 32 \tag{C.10}$$

Now that we have derived a high confidence bound for W_h we can obtain a bound for its expected value:

$$\mathbb{E}\left[W_h\right] = \sum_{i=1}^{16\log n} \mathbf{Pr}\left(W_h \geq i\right) + \sum_{i>16\log n} \mathbf{Pr}\left(W_h \geq i\right) \leq 16\log n + c = \mathcal{O}(\log n) \tag{C.11}$$

Observe that the first sum is bounded above $16\log n$ because every probability is less than 1 and the second one is dominated by $\sum_{i\geq 1} 1/i^2$ which is a constant. Since the running time of a search operation is bounded by the number of horizontal moves performed at each level plus the number of vertical moves to reach the target node from the topmost level to the bottom most. We have that the overall running time of a search procedure starting at the left topmost node in the skip list is $T = h + W_h = \mathcal{O}(\log n)$ with probability at least $1 - (1/n^2)$, by applying the union bound we have that T is $\mathcal{O}(\log n)$ with probability at least $1 - (1/n)$ starting at any node in the skip list. To

conclude, the expected running time is $\mathbb{E}[T] = \mathbb{E}[h] + \mathbb{E}[W_h] = \mathcal{O}(\log n)$. □

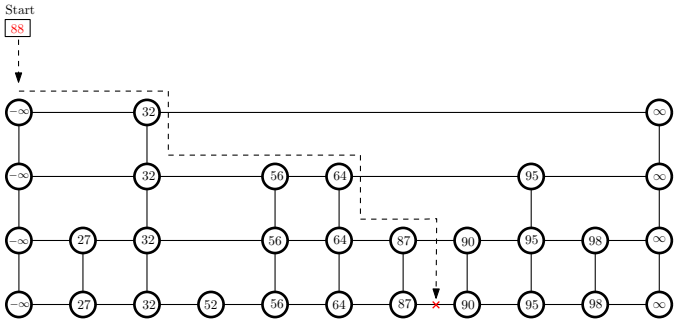


FIGURE C.1: Example of the insertion of the element 88 in the skip list. Dashed line is the search path.

C.2 Spartan's Reshaping protocol

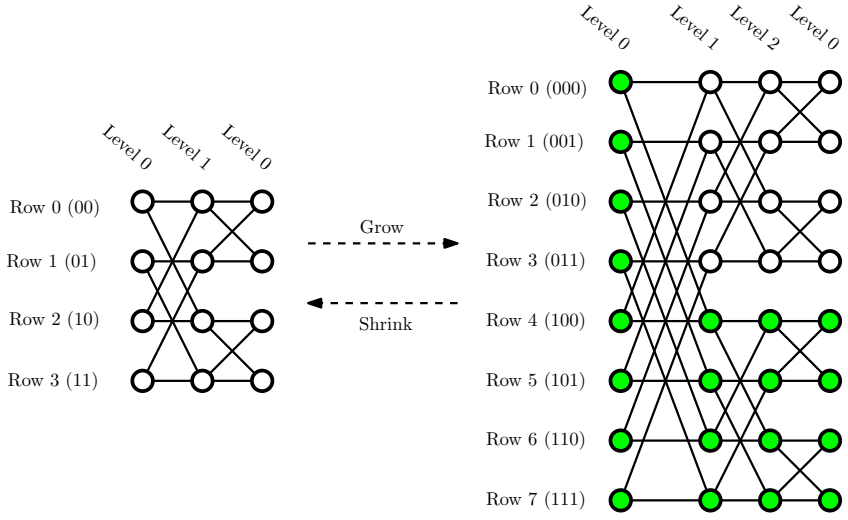


FIGURE C.2: Example of the reshaping procedure. On the left, the Spartan structure as a one-dimensional wrapped butterfly of committees. The rows are represented with their binary representations. Every node in the figure is a random committee of $\Theta(\log n)$ nodes and every edge between two committees encodes a complete bipartite graph. On the right, 2-dimensional butterfly is obtained by increasing the dimensionality of the left-wrapped butterfly by one. Green nodes are the ones that must be added to increase the dimensionality of the 1-dimensional wrapped butterfly. Moreover, left-to-right execution increases the dimensionality by one, and right-to-left execution decreases the dimensionality by one.

In this section, we provide a detailed description of the reshaping protocol mentioned in Section 6.2.1. Let $\alpha, \beta \in \mathbb{R}_{\geq 0}$ such that $\alpha > 1$ and $\beta \in (0, 1)$ be two constants. We want the Spartan network $\mathcal{S}$ to be enlarged if all the committees have size at most $\alpha c \log n$, to shrink if all the committees have size at most $\beta c \log n$, or stay the same otherwise ($c > 0$ is the constant hidden by the asymptotic notation). In other words, we want $\mathcal{S}$ to be enlarged or reduced when the wrapped butterfly network size increases or

decreases by one dimension respectively. Moreover, a general technique to increase the dimensionality of a k-dimensional butterfly network $\mathcal{S}$ by one is to: (1) create a copy of $\mathcal{S}$ in which each row ID (i, j) is "shifted" by 2^k i.e., $(2^k + i, j)$; (2) "shift" the index of the levels of the two disjoint butterflies by one; (3) add a new row of 2^{k+1} nodes to form the level 0 of the $k + 1$ dimensional butterfly; and, (5) connect the newly created level 0 to the two disjoint butterflies using the "standard" butterfly edge-construction rule [181]. Furthermore, to reduce the dimensionality of a k-dimensional butterfly, we need to remove the level 0, remove one of the two disjoint $k - 1$-dimensional butterflies, and fix the row and levels IDs if needed. Figure C.2 shows an example on how to perform such operations. We start with the description of the growing procedure. A new number of nodes $n' = \alpha n$ is reached such that $2^k k \in \mathcal{O}(N)$ must be increased to $2^{k+1}(k + 1) \in \mathcal{O}(N')$ (the number of committees increases from $N = \mathcal{O}(n/\log n)$ to $N' = \mathcal{O}(n'/\log n')$) and update the butterfly structure according to the new size of the network. Moreover, each step of Algorithm 6 requires at most $\mathcal{O}(\log n)$ rounds and no node will send or receive more than $\mathcal{O}(\log n)$ messages at any round. As a first step, the committee $C(0, 0)$ is elected as a leader[1], and each committee routes a message in which it expresses its opinion (grow, shrink, stay the same) to the leader committee (lines 1-2). This can be done very efficiently on hypercubic networks [39, 181], indeed we have that the leader committee will receive all the opinions in $\mathcal{O}(\log N)$ rounds [196] where $N \in \mathcal{O}(n/\log n)$ is the number of committees in the wrapped butterfly. If the network agrees on growing, the protocol proceeds with increasing the dimensionality of the wrapped butterfly by one. Moreover, assume that the k-dimensional butterfly must be transformed into a $k + 1$-dimensional one. To this end, the protocol executes the growing approach described before. Every committee shifts its level ID by one and promotes two random nodes in each committee to be the committee leaders of the copy of the butterfly and of the new level 0. Next, all the newly elected committee leaders will start recruiting random nodes in the network until they reach a committee size of $\Theta(\log n')$ where $n' = \alpha n$. Finally, each new committee will create edges according to the wrapped butterfly construction algorithm [14, 181] and drop the old ones (if any). The next lemma shows that the growing phase requires $\mathcal{O}(\log n')$ rounds w.h.p. where $n' = \alpha n$.

Lemma C.4. *The network growth process ensures that every committee has* $\Theta(\log n')$ *members after* $\mathcal{O}(\log n')$ *rounds w.h.p.*

Proof. Let $b = \frac{3n'}{4N'} = \frac{3}{4}\log n'$ we show that after the first phase of the process, each new committee with gain b nodes. During each round, there are at least $\frac{n'}{4} - N' \in \Omega(n')$ nodes that did not receive any invitation from one of these new committee leaders.

[1] We can make this assumption because we do not deal with byzantine nodes.

Moreover, define the event E_v = "The committee leader v recruits a node" then setting $N' = \frac{n'}{c \log n'}$ with $c \geq 1$ gives $\mathbf{Pr}(E_v) \geq \left(\frac{n'/4 - N'}{n'}\right)\left(1 - \frac{1}{n'}\right)^{N'} \geq \frac{3}{16}$. To bound the expected time needed by a committee to recruit b elements we define a pure-birth Markov Chain $\{X_t\}_{t \geq 0}$ with state space $\Omega = \{0, 1, 2, \ldots, b\}$ and initial state 0 that counts the number of recruited members. At each round, the Markov Chain at state $i < b$ can proceed one step forward to state $i + 1$ with probability $p_{i,i+1} = \mathbf{Pr}(E_v)$ or loop on i with probability $r_{i,i} = 1 - p_{i,i+1}$. Observe that each state $0 \leq i < b$ is a transient state, while b is an absorbing state, and that the expected absorption time in state b is $\mathcal{O}(b) = \mathcal{O}(\log n')$. To give a tail bound on the absorption time, we study a more pessimistic random process in which we toss a p-biased coin with $p = 3/16$ and count the number of rounds Z_i before we get b heads. In this experiment, the expected number of rounds needed to get b heads is $\frac{16}{3} b \in \Theta(\log n')$. By applying a Chernoff bound (Theorem 2.14) we can show that for any value of n' and any fixed R, such that $R \geq 6\mathbb{E}[Z]$ the probability that we will need more than $R \log n'$ rounds to get b number of heads is at most $1/n'^R$. In other words, this means that the Markov Chain will take at most $\mathcal{O}(\log n')$ rounds with probability $1 - 1/n'^R$ to reach state b. Finally, by applying the union bound we can show that every committee will need $\mathcal{O}(\log n')$ rounds with probability $1 - 1/n'^{R-1}$ to recruit b members. □

Furthermore, in the case in which the network agreed on shrinking i.e., reducing the butterfly dimensionality from k to $k-1$ a pool of vacated nodes $\mathcal{V}$ is created and all the nodes in the level 0 and in the committees $C(i, j)$ such that $2^{k-1} \leq i < 2^k$ and $1 \leq j \leq k$ join such pool of nodes. Next, each committee in the remaining butterfly network will "recruit" $b = \frac{3}{4} \log n'$ (here $n' = \beta n$) random nodes from $\mathcal{V}$. After the recruitment phase, each remaining node u in $\mathcal{V}$ (if any) will probe a random committee in the butterfly. If such a committee has not reached the target size yet it accepts the request, otherwise u tries again with a different committee. It follows that the shrinking phase can be done in $\mathcal{O}(\log n')$ rounds w.h.p. where $n' = \beta n$.

Lemma C.5. *The network shrinking process ensures that every committee has* $\Theta(\log n')$ *members after* $\mathcal{O}(\log n')$ *rounds w.h.p.*

Proof. Let $b = \frac{3n'}{4N'} = \frac{3}{4} \log n'$ and observe that the analysis of the recruitment phase is identical to the one for the growing process. Moreover, after $\Theta(\log n)$ rounds, there can be $n'/4$ nodes in $\mathcal{V}$ that are not part of any committee. Each node that joined a committee has one unit of budget that can use to induce a new member into its committee. Every remaining node $u \in \mathcal{V}$ probes a random node v asking to join its committee, if v's budget is greater than 0 then v accepts the request, otherwise u tries again with a different node. Furthermore, observe that even if all the remaining nodes $\mathcal{V}$

found a spot in some committee, there will be $n'/2$ nodes that would not have exhausted their budget and each remaining node can find a committee with probability at least $1/2$. Finally, by using similar arguments to the growing process we can show that each remaining node u succeed in finding a committee with high probability in $\Theta(\log n')$ rounds. And by using the union bound we can guarantee that each remaining node can find and become part of a committee in $\Theta(\log n')$ w.h.p. □

Algorithm 6: Reshape protocol.

Agreement Phase.

The committee $C(0,0)$ is elected as a leader ℓ.
Every committee in $\mathcal{S}$ routes its opinion to ℓ.
if *Agreement on Growing* **then**
    Growing Phase. // Here $n' = \alpha n$
    Each committee $C(i,j)$ for $0 \leq i < 2^k$ and $0 \leq j < k$:
        (a) Increases its committee ID from $C(i,j)$ to $C(i,j+1)$.
        (b) Promotes a random node u among the available committee members to be the leader of the new committee $C(j \cdot 2^{k-1} + i, 0)$.
        (c) Promotes a random node v among the available committee members to be the leader of the new committee $C(2^{k-1} + i, j+1)$.
    Each newly generated committee leader actively recruits new nodes in their committee until it gets $\Theta(\log n')$ committee members.
    Each new committee creates complete bipartite edges according to the Wrapped Butterfly construction algorithm (see [14, 181]).
else if *Agreement on Shrinking* **then**
    Shrinking Phase. // Here $n' = \beta n$
    Each member in the committees $C(i,0)$ for $0 \leq i < 2^k$ and $C(i,j)$ for $2^{k-1} \leq i < 2^k$ $1 \leq j \leq k$ "vacate" its committee and joins a pool of unassigned nodes.
    Each node in the pool of unassigned nodes will randomly join one of the $N' = \mathcal{O}(n'/\log n')$ new committees such that each committee has $\Theta(\log n')$ nodes.
else
    Do nothing.

Bibliography

[1] Paul Liu, Austin R. Benson, and Moses Charikar. Sampling methods for counting temporal motifs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, WSDM 2019, Melbourne, VIC, Australia, February 11-15, 2019*. ACM, 2019.

[2] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 1977.

[3] Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. Algorithmic aspects of temporal betweenness. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2020.

[4] Diego Santoro and Ilie Sarpe. ONBRA: rigorous estimation of the temporal betweenness centrality in temporal networks. *CoRR*, 2022.

[5] Martin G. Everett and Stephen P. Borgatti. Ego network betweenness. *Soc. Networks*, 27(1):31–38, 2005. doi: 10.1016/j.socnet.2004.11.007. URL https://doi.org/10.1016/j.socnet.2004.11.007.

[6] Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Finding a bounded-degree expander inside a dense one. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. SIAM, 2020.

[7] James Aspnes and Gauri Shah. Skip graphs. *ACM Trans. Algorithms*, 2007.

[8] 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.

[9] 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.

[10] Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip$^+$: A self-stabilizing skip graph. *J. ACM*, 2014.

[11] 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.

[12] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. DEX: self-healing expanders. *Distributed Comput.*, 2016.

[13] Maximilian Drees, Robert Gmyr, and Christian Scheideler. Churn- and dos-resistant overlay networks based on network reconfiguration. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*. ACM, 2016.

[14] John Augustine and Sumathi Sivasubramaniam. Spartan: Sparse robust addressable networks. *J. Parallel Distributed Comput.*, 2021.

[15] John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine agreement in dynamic networks. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*. ACM, 2013.

[16] John Augustine, Gopal Pandurangan, and Peter Robinson. Fast byzantine leader election in dynamic networks. In *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*. Springer, 2015.

[17] John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Storage and search in dynamic peer-to-peer networks. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*. ACM, 2013.

[18] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 2001.

[19] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2015.

[20] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM*

Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021. SIAM, 2021.

[21] Maciej Rymar, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Towards classifying the polynomial-time solvability of temporal betweenness centrality. In *Graph-Theoretic Concepts in Computer Sciences*, Lecture Notes in Computer Science. Springer, 2021.

[22] Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. Graph metrics for temporal networks. In *Temporal networks*. Springer, 2013.

[23] Pierluigi Crescenzi, Clémence Magnien, and Andrea Marino. Finding top-k nodes for temporal closeness in large temporal graphs. *Algorithms*, 2020.

[24] Lutz Oettershagen and Petra Mutzel. Efficient top-k temporal closeness calculation in temporal networks. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020.

[25] Shahrzad Haddadan, Cristina Menghini, Matteo Riondato, and Eli Upfal. Repbublik: Reducing polarized bubble radius with link insertions. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*. ACM, 2021.

[26] John Kit Tang, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Vincenzo Nicosia. Analysing information flows and key mediators through temporal centrality metrics. In *Proceedings of the 3rd Workshop on Social Network Systems*, 2010.

[27] John Kit Tang, Cecilia Mascolo, Mirco Musolesi, and Vito Latora. Exploiting temporal complex network metrics in mobile malware containment. In *12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WOWMOM 2011, Lucca, Italy, 20-24 June, 2011*. IEEE Computer Society, 2011.

[28] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 1953.

[29] Peter Grindrod, Mark C Parsons, Desmond J Higham, and Ernesto Estrada. Communicability across evolving networks. *Physical Review E*, 2011.

[30] Ferenc Béres, Róbert Pálovics, Anna Oláh, and András A Benczúr. Temporal walk based centrality metric for graph streams. *Applied network science*, 2018.

[31] Polina Rozenshtein and Aristides Gionis. Temporal pagerank. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016.

[32] Ioanna Tsalouchidou, Ricardo Baeza-Yates, Francesco Bonchi, Kewen Liao, and Timos Sellis. Temporal betweenness centrality in dynamic graphs. *Int. J. Data Sci. Anal.*, 2020.

[33] Matteo Riondato and Eli Upfal. ABRA: approximating betweenness centrality in static and dynamic graphs with rademacher averages. *ACM Trans. Knowl. Discov. Data*, 2018.

[34] Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample-variance penalization. In *COLT The 22nd Conference on Learning Theory*, 2009.

[35] Marwan Ghanem, Florent Coriat, and Lionel Tabourier. Ego-betweenness centrality in link streams. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, Sydney, Australia, July 31 - August 03, 2017*. ACM, 2017.

[36] Frédéric Simard, Clémence Magnien, and Matthieu Latapy. Computing betweenness centrality in link streams. *CoRR*, 2021.

[37] Lutz Oettershagen, Petra Mutzel, and Nils M. Kriege. Temporal walk centrality: Ranking nodes in evolving networks. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*. ACM, 2022.

[38] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 1963.

[39] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[40] Sheldon M Ross, Sheldon M Ross, Sheldon M Ross, and Sheldon M Ross. *A first course in probability*. Macmillan New York, 1976.

[41] Shang-Hua Teng. Scalable algorithms for data and network analysis. *Found. Trends Theor. Comput. Sci.*, 2016.

[42] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. In *Proceedings of the 16th Italian Conference on Theoretical Computer Science*, 2015.

[43] R. Goerke. Email network of KIT informatics. https://i11www.iti.kit.edu/en/projects/spp1307/emaildata, 2011. Online; accessed 10 February 2023.

[44] Jure Leskovec and Andrej Krevl. Snap datasets. http://snap.stanford.edu/data.

[45] Michele Borassi and Emanuele Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. *ACM J. Exp. Algorithmics*, 2019.

[46] Leonardo Pellegrina and Fabio Vandin. Silvan: Estimating betweenness centralities with progressive sampling and non-uniform rademacher bounds. *ACM Trans. Knowl. Discov. Data*, 2023.

[47] Sociopatterns. https://www.sociopatterns.org/.

[48] Ryan A. Rossi and Nesreen K. Ahmed. Network repository. https://networkrepository.com.

[49] J. Kunegis. The KONECT Project. http://konect.cc.

[50] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, 2009.

[51] The SciPy community. Statistical functions. https://docs.scipy.org/doc/scipy/reference/stats.html, last checked on February 10, 2023.

[52] Sebastiano Vigna. A weighted correlation index for rankings with ties. In *Proceedings of the 24th international conference on World Wide Web*, 2015.

[53] Charles Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, 1904.

[54] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 1938.

[55] Leonardo Pellegrina. Efficient centrality maximization with rademacher averages. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*. ACM, 2023.

[56] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 2003.

[57] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 2003.

[58] David Pollard. *Convergence of stochastic processes*. Springer Science & Business Media, 2012.

[59] Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Trans. Inf. Theory*, 2001.

[60] Leonardo Pellegrina, Cyrus Cousins, Fabio Vandin, and Matteo Riondato. Mcrapper: Monte-carlo rademacher averages for poset families and approximate pattern mining. *ACM Trans. Knowl. Discov. Data*, 2022.

[61] S Boucheron, G Lugosi, and P Massart. Concentration inequalities: A nonasymptotic theory of independence. univ. press, 2013.

[62] Olivier Bousquet. A bennett concentration inequality and its application to suprema of empirical processes. *Comptes Rendus Mathematique*, 2002.

[63] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity: festschrift for alexey chervonenkis*. Springer, 2015.

[64] Sariel Har-Peled and Micha Sharir. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry*, 2011.

[65] Yi Li, Philip M Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 2001.

[66] Maarten Löffler and Jeff M Phillips. Shape fitting on point sets with probability distributions. In *European symposium on algorithms*. Springer, 2009.

[67] Rajendra Bhatia and Chandler Davis. A better bound on the variance. *Am. Math. Mon.*, 2000.

[68] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., USA, 1990.

[69] Leonardo Pellegrina. Rigorous and efficient algorithms for significant and approximate pattern mining. *Ph.D. Thesis https://hdl.handle.net/11577/3471458*, 2021.

[70] Marco Calamai, Pierluigi Crescenzi, and Andrea Marino. On computing the diameter of (weighted) link streams. *ACM J. Exp. Algorithmics*, 2022.

[71] Pierluigi Crescenzi, Clémence Magnien, and Andrea Marino. Approximating the temporal neighbourhood function of large temporal graphs. *Algorithms*, 2019.

[72] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 2014.

[73] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. 2016.

[74] Foster J. Provost, David D. Jensen, and Tim Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1999.

[75] R. Kujala, C. Weckström, R. Darst, M. Madlenocić, and J. Saramäki. A collection of public transport network data sets for 25 cities. *Sci. Data*, 2018.

[76] Ruben Becker, Pierluigi Crescenzi, Antonio Cruciani, and Bojana Kodric. Proxying betweenness centrality rankings in temporal networks. In *21st International Symposium on Experimental Algorithms, SEA*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[77] Cyrus Cousins, Chloe Wohlgemuth, and Matteo Riondato. Bavarian: Betweenness centrality approximation with variance-aware rademacher averages. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

[78] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[79] Ivan Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216, Amsterdam, 1987. Springer. doi: 10.1007/3-540-39118-5_19. URL https://doi.org/10.1007/3-540-39118-5_19.

[80] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[81] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, 1992. Springer. doi: 10.1007/3-540-48071-4_10. URL https://doi.org/10.1007/3-540-48071-4_10.

[82] David Barkai. *Technologies for Sharing and Collaborating on the Net*. IEEE Computer Society, Linköping, 2001. doi: 10.1109/P2P.2001.990419. URL https://doi.org/10.1109/P2P.2001.990419.

[83] Satoshi Nakamoto and et Al. Bitcoin core. https://github.com/bitcoin/bitcoin, 2008. Accessed: 2022-07-22.

[84] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the open blockchain.* O'Reilly Media, Inc., 2nd edition, 2017. ISBN 1491954388.

[85] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, Princeton, 2016.

[86] Bitcoin Core. Bitcoin Core 0.11 (ch 4): P2P Network. https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_4):_P2P_Network, 2022. Accessed: 2022-07-22.

[87] Addy Yeow. Global Bitcoin Nodes Distribution. https://bitnodes.io/, 2013. Accessed: 2022-07-22.

[88] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. Txprobe: Discovering bitcoin's network topology using orphan transactions. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 550–566, Frigate Bay, 2019. Springer. doi: 10.1007/978-3-030-32101-7_32. URL https://doi.org/10.1007/978-3-030-32101-7_32.

[89] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. Timing analysis for inferring the topology of the bitcoin peer-to-peer network. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*, pages 358–367, Toulouse, 2016. IEEE Computer Society. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070. URL https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070.

[90] Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Finding a bounded-degree expander inside a dense one. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1320–1336, Salt Lake City, 2020. SIAM. doi: 10.1137/1.9781611975994.80. URL https://doi.org/10.1137/1.9781611975994.80.

[91] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. Erlay: Efficient transaction relay for bitcoin. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 817–831, London, 2019. ACM. doi: 10.1145/3319535.3354237. URL https://doi.org/10.1145/3319535.3354237.

[92] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *Proc. ACM Meas. Anal. Comput. Syst.*, 2(2):29:1–29:35, 2018. doi: 10.1145/3224424. URL https://doi.org/10.1145/3224424.

[93] Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys & Tutorials*, 21(1):838–857, 2018.

[94] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering bitcoin's public topology and influential nodes. *et al*, 2015.

[95] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. *IEEE Journal on selected areas in communications*, 21(6): 995–1002, 2003. Preliminary version in FOCS'01.

[96] 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.

[97] 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 Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 350–369, Berkeley, 2015. IEEE Computer Society. doi: 10.1109/FOCS.2015.29. URL https://doi.org/10.1109/FOCS.2015.29.

[98] Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Sampling regular graphs and a peer-to-peer network. *Comb. Probab. Comput.*, 16(4):557–593, 2007. doi: 10.1017/S0963548306007978. URL https://doi.org/10.1017/S0963548306007978.

[99] Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. The effect of faults on network expansion. *Theory of Computing Systems*, 39(6):903–928, 2006. Preliminary version in SPAA'04.

[100] Luca Becchetti, Andrea E. F. Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi. Expansion and flooding in dynamic random networks with node churn. In *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*, pages 976–986, Washington DC, 2021. IEEE. doi: 10.1109/ICDCS51616.2021.00097. URL https://doi.org/10.1109/ICDCS51616.2021.00097.

[101] John Augustine, Gopal Pandurangan, and Peter Robinson. Distributed algorithmic foundations of dynamic networks. *SIGACT News*, 2016.

[102] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003. doi: 10.1145/945506.945507. URL https://doi.org/10.1145/945506.945507.

[103] Giulia Fanti and Pramod Viswanath. Deanonymization in the bitcoin P2P network. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1364–1373, Long Beach, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/6c3cf77d52820cd0fe646d38bc2145ca-Abstract.html.

[104] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1):22:1–22:34, 2017. doi: 10.1145/3084459. URL https://doi.org/10.1145/3084459.

[105] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust P2P system to handle flash crowds. *IEEE J. Sel. Areas Commun.*, 22(1):6–17, 2004. doi: 10.1109/JSAC.2003.818778. URL https://doi.org/10.1109/JSAC.2003.818778.

[106] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: inexpensive membership management for unstructured P2P overlays. *J. Netw. Syst. Manag.*, 13(2):197–217, 2005. doi: 10.1007/s10922-005-4441-x. URL https://doi.org/10.1007/s10922-005-4441-x.

[107] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*,

25(3):8, 2007. doi: 10.1145/1275517.1275520. URL https://doi.org/10.1145/1275517.1275520.

[108] James R. Norris. *Markov chains*. Cambridge university press, Cambridge, 1998.

[109] David Aldous and James Allen Fill. Reversible Markov Chains and Random Walks on Graphs, 2002. Unfinished monograph, recompiled 2014, available at http://www.stat.berkeley.edu/$\sim$aldous/RWG/book.html.

[110] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[111] Salil P Vadhan et al. *Pseudorandomness*. Now Publishers, Inc., 2012.

[112] Brian Everitt and Anders Skrondal. *The Cambridge dictionary of statistics*, volume 106. Cambridge university press Cambridge, Cambridge, 2002.

[113] Liang Wang and Ivan Pustogarov. Towards better understanding of bitcoin unreachable peers. *CoRR*, abs/1709.06837, 2017. URL http://arxiv.org/abs/1709.06837.

[114] Inc Wikipedia. Wikipedia page about gnutella. https://en.wikipedia.org/wiki/Gnutella, last checked on 14-February-2024.

[115] Inc Microsoft. Website of skype. https://www.skype.com/, last checked on 14-February-2024.

[116] Inc BitTorrent. Website of bittorrent. https://www.bittorrent.com/, last checked on 14-February-2024.

[117] Inc Crashplan. Website of crashplan. https://crashplan.com/, last checked on 14-February-2024.

[118] Inc Sysform. Website of sysform. https://symform.com/, last checked on 14-February-2024.

[119] Inc Signal. Website of signal. https://signal.org/, last checked on 14-February-2024.

[120] 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.

[121] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *Comput. Commun. Rev.*, 2002.

[122] Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw.*, 2004.

[123] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference, IMC 2006, Rio de Janeriro, Brazil, October 25-27, 2006*. ACM, 2006.

[124] Arjan Peddemors. Cloud storage and peer-to-peer storage-end-user considerations and product overview. *GigaPort3 deliverable*, 2010.

[125] 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.

[126] 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.

[127] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy H. Campbell. A survey of peer-to-peer storage techniques for distributed file systems. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 2, 4-6 April 2005, Las Vegas, Nevada, USA*. IEEE Computer Society, 2005.

[128] 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.

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

[130] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Comput.*, 2006.

[131] Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. Security applications of peer-to-peer networks. *Comput. Networks*, 2004.

[132] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proceedings of the 2005 ACM Workshop on Rapid Malcode, WORM 2005, Fairfax, VA, USA, November 11, 2005*. ACM Press, 2005.

[133] 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.

[134] Özalp Babaoglu, Moreno Marzolla, and Michele Tamburini. Design and implementation of a P2P cloud system. In *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012*. ACM, 2012.

[135] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001.

[136] Ion Stoica, Robert Tappan Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 2003.

[137] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *Comput. Commun. Rev.*, 2002.

[138] 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.

[139] Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In *SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, July 18-20, 2005, Las Vegas, Nevada, USA*. ACM, 2005.

[140] Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, and Christian Sohler. Distributed monitoring of network properties: The power of hybrid networks. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[141] Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, Lecture Notes in Computer Science. Springer, 2019.

[142] Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*. ACM, 2021.

[143] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 1983.

[144] Soumyottam Chatterjee, Gopal Pandurangan, and Nguyen Dinh Pham. Distributed MST: A smoothed analysis. In *ICDCN 2020: 21st International Conference on Distributed Computing and Networking, Kolkata, India, January 4-7, 2020*. ACM, 2020.

[145] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM J. Comput.*, 1982.

[146] Eli Upfal. An o(log(n)) deterministic packet-routing scheme. *J. ACM*, 1992.

[147] 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.

[148] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 2022.

[149] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 1990.

[150] Hagit Attiya, Sweta Kumari, Archit Somani, and Jennifer L. Welch. Store-collect in the presence of continuous churn with application to snapshots and lattice agreement. *Inf. Comput.*, 2022.

[151] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*. ACM, 1983.

[152] Kenneth E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968.

[153] Baruch Awerbuch and Christian Scheideler. The hyperring: a low-congestion deterministic data structure for distributed environments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*. SIAM, 2004.

[154] Ankur Bhargava, Kishore Kothapalli, Chris Riley, Christian Scheideler, and Mark Thober. Pagoda: a dynamic overlay network for routing, data management, and multicasting. In *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*. ACM, 2004.

[155] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002*. ACM, 2002.

[156] Moni Naor and Udi Wieder. Novel architectures for P2P applications: The continuous-discrete approach. *ACM Trans. Algorithms*, 2007.

[157] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA*. ACM, 2001.

[158] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter P2P networks. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001.

[159] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Comput.*, 2010.

[160] Tim Jacobs and Gopal Pandurangan. Stochastic analysis of a churn-tolerant structured peer-to-peer scheme. *Peer-to-Peer Netw. Appl.*, 2013.

[161] Luca Becchetti, Andrea E. F. Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi. Expansion and flooding in dynamic random networks with node churn. *Random Struct. Algorithms*, 2023.

[162] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001,*

IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings, Lecture Notes in Computer Science. Springer, 2001.

[163] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: the design of a large-scale event notification infrastructure. In *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*, Lecture Notes in Computer Science. Springer, 2001.

[164] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, Lecture Notes in Computer Science. Springer, 2003.

[165] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed hashing in a small world. In *4th USENIX Symposium on Internet Technologies and Systems, USITS'03, Seattle, Washington, USA, March 26-28, 2003*. USENIX, 2003.

[166] Prasanna Ganesan and Gurmeet Singh Manku. Optimal routing in chord. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*. SIAM, 2004.

[167] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Commun. Surv. Tutorials*, 2005.

[168] Thomas Papadakis, J. Ian Munro, and Patricio V. Poblete. Analysis of the expected search cost in skip lists. In *SWAT 90, 2nd Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 11-14, 1990, Proceedings*, Lecture Notes in Computer Science. Springer, 1990.

[169] Luc Devroye. A limit theory for random skip lists. *The Annals of Applied Probability*, 1992.

[170] Peter Kirschenhofer and Helmut Prodinger. The path length of random skip lists. *Acta Informatica*, 1994.

[171] Peter Kirschenhofer, Conrado Martinez, and Helmut Prodinger. Analysis of an optimized search algorithm for skip lists. *Theor. Comput. Sci.*, 1995.

[172] Joaquim Gabarró Vallès, Conrado Martínez Parra, and Xavier Messeguer Peypoch. A top-down design of a parallel dictionary using skip lists. *Theoretical Computer Science*, 1994.

[173] Joaquim Gabarró and Xavier Messeguer. A unified approach to concurrent and parallel algorithms on balanced data structures. In *Proceedings 17th International Conference of the Chilean Computer Science Society*. IEEE, 1997.

[174] William Pugh. Concurrent maintenance of skip lists. In *Technical Report*, 1998.

[175] Nir Shavit and Itay Lotan. Skiplist-based concurrent priority queues. In *Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, May 1-5, 2000*. IEEE Computer Society, 2000.

[176] Håkan Sundell and Philippas Tsigas. Scalable and lock-free concurrent dictionaries. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*. ACM, 2004.

[177] Keir Fraser. Practical lock-freedom. Technical report, University of Cambridge, Computer Laboratory, 2004.

[178] Maurice Herlihy, Yossi Lev, Victor Luchangco, and Nir Shavit. A provably correct scalable concurrent skip list. In *Conference On Principles of Distributed Systems (OPODIS). Citeseer*, volume 103, 2006.

[179] James Aspnes and Udi Wieder. The expansion and mixing time of skip graphs with applications. *Distributed Comput.*, 2009.

[180] Shlomi Dolev. Self stabilization. *Journal of Aerospace Computing, Information, and Communication*, 2004.

[181] F Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays· trees· hypercubes*. Elsevier, 2014.

[182] John Augustine and Sumathi Sivasubramaniam. Spartan: A framework for sparse robust addressable networks. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*. IEEE Computer Society, 2018.

[183] John Augustine, Keerti Choudhary, Avi Cohen, David Peleg, Sumathi Sivasubramaniam, and Suman Sourav. Distributed graph realizations. *IEEE Trans. Parallel Distributed Syst.*, 2022.

[184] Mike Paterson. Improved sorting networks with o(log N) depth. *Algorithmica*, 1990.

[185] Bruce M. Maggs and Berthold Vöcking. Improved routing and sorting on multibutterflies. *Algorithmica*, 2000.

[186] John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*. ACM, 2019.

[187] John Kit Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Temporal distance metrics for social network analysis. In *Proceedings of the 2nd ACM Workshop on Online Social Networks, WOSN 2009, Barcelona, Spain, August 17, 2009*. ACM, 2009.

[188] Leonardo Maccari, Lorenzo Ghiro, Alessio Guerrieri, Alberto Montresor, and Renato Lo Cigno. On the distributed computation of load centrality and its application to DV routing. In *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018.

[189] Tianming Zhang, Yunjun Gao, Jie Zhao, Lu Chen, Lu Jin, Zhengyi Yang, Bin Cao, and Jing Fan. Efficient exact and approximate betweenness centrality computation for temporal graphs. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore*. ACM, 2024.

[190] Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. Making temporal betweenness computation faster and restless. In *To appear in KDD 2024*. ACM, 2024.

[191] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 2002.

[192] Lutz Oettershagen and Petra Mutzel. An index for temporal closeness computation in evolving graphs. In *Proceedings of the 2023 SIAM International Conference on Data Mining, SDM 2023*. SIAM, 2023.

[193] Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *Int. J. Bifurc. Chaos*, 2007.

[194] Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. In *Network Analysis: Methodological Foundations*. Springer, 2004.

[195] Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *Seventh ACM International Conference on Web Search and Data Mining, WSDM*. ACM, 2014.

[196] Frank Thomson Leighton, Bruce M. Maggs, Abhiram G. Ranade, and Satish Rao. Randomized routing and sorting on fixed-connection networks. *J. Algorithms*, 1994.