ACHIEVING EFFICIENCY, ROBUSTNESS, AND SECURITY IN DISTRIBUTED COMPUTING

A Dissertation Presented to

the Faculty of the Department of Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Soumyottam Chatterjee

August 2019

ACHIEVING EFFICIENCY, ROBUSTNESS, AND SECURITY IN DISTRIBUTED COMPUTING

Soumyottam Chatterjee, Author and Candidate

APPROVED:

Gopal Pandurangan, Chairman
Professor, Department of Computer Science
University of Houston

Rakesh Verma
Professor, Department of Computer Science
University of Houston

Lennart Johnsson
Professor, Department of Computer Science
University of Houston

Peter Robinson
Assistant Professor, Department of Computer Science
City University of Hong Kong

Dan E. Wells
Professor and Dean
College of Natural Sciences and Mathematics
University of Houston

Acknowledgments

First and foremost, with great pleasure, I express my heartfelt gratitude to my supervisor Professor Gopal Pandurangan for his guidance and support throughout my time in UH. His amazing way of explanation during the analysis of algorithms always makes me feel excited to be involved with the problem. Also I want to mention his excellent teaching, especially his course on distributed algorithms, which helped a lot. Above all, he provided me unflinching encouragement and support in various ways. I will forever be indebted to him for providing me with the opportunity to work under his supervision and for his invaluable guidance.

I would like to thank the National Science Foundation (NSF) for their research grants that helped finance my research. In particular, the following NSF grants have sponsored my research: CCF- 1527867, CCF-1540512, IIS-1633720, and CCF-1717075.

I am extremely grateful to my collaborators. Especially, a large fraction of my gratitude goes to my coauthor Dr. Peter Robinson for providing me with a wonderful research experience from the first stage. It is fun to work with Peter. I am also thankful to Dr. John Augustine and Dr. Robert Gmyr for their help and encouragement. I would like to thank (soon-to-be-doctors) Mr. Reza Fathi and Mr. Nguyen Dinh Pham, whom I could find beside me whenever I needed them.

ACHIEVING EFFICIENCY, ROBUSTNESS, AND SECURITY IN DISTRIBUTED COMPUTING

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Soumyottam Chatterjee

August 2019

Abstract

This dissertation investigates ways to improve various performance metrics of distributed computing systems. We begin with a study of the *message complexity* of the *leader election* problem in diameter-two networks. We first give a simple randomized Monte-Carlo leader election algorithm that with high probability[1] succeeds and uses $O(n\log^3 n)$ messages and runs in $O(1)$ rounds. We then show that any algorithm (even Monte Carlo randomized algorithms with large enough constant success probability) needs $\Omega(n)$ messages (even when n is known), regardless of the number of rounds. We also present an $O(n\log n)$ message deterministic algorithm that takes $O(\log n)$ rounds (but needs knowledge of n); we show that this message complexity is tight for deterministic algorithms.

Next we move on to designing algorithms that are provably robust to network malfunction and instability. First we consider the *Byzantine counting* problem, that asks for an estimate of the network size n (i.e., the number of nodes in the network) in the presence of Byzantine or malicious nodes. We design a fast (running in $O(\log^3 n)$ rounds) algorithm that guarantees that most nodes in the network (i.e., $\geq (1-\varepsilon)$-fraction of the nodes; for any small $\varepsilon > 0$) will know a constant factor approximation of $\log n$. Ours is the first fully local (decentralized and needing no global information) distributed algorithm that solves this important problem.

[1] Throughout this dissertation, we would use "eith high probability" or "whp" to mean "with probability at least $1-n^{-c}$, where n is the network size (i.e., the number of nodes in the network) and c is some fixed positive constant.

Finally we design and analyze a randomized distributed protocol that guarantees with high probability the construction and maintenance of a sparse (where each node in the network has degree at most $\text{polylog}(n)$) distributed hash table (DHT) network that guarantees efficient and reliable communication between (almost all pairs of) good nodes even in the presence of $\frac{n}{\text{polylog}(n)}$ number of Byzantine nodes and *continual heavy* churn of up to $\frac{n}{\text{polylog}(n)}$ *adversarially* chosen nodes joining and leaving every round (where n is the stable network size). To our knowledge, this is the first protocol that can guarantee the above properties under such a large number of Byzantine nodes in a highly dynamic setting.

Contents

This dissertation is dedicated to

Professor Arijit Bishnu

for standing by me when no one else would,

and

Professor Gopal Pandurangan[2]

for believing in me when no one else would.

[2]Please note that the names are listed alphabetically, as is the custom in the TCS community.

Chapter 1

Introduction

This dissertation studies *distributed computing* and, in particular, investigates ways to improve various performance metrics of distributed computing systems. The very first problem that we investigate concerns the performance metric called *message complexity* (see Section 1.1.2 for the definition of "message complexity"). We study the message complexity of the "leader election" problem. In the "leader election" problem, the goal is to elect a unique leader of the entire network. That is, at the end of leader election, exactly one process would know that it is the leader; all others would know that they are not. Leader election is a basic primitive that serves in many applications. Please see Chapter 2 (Section 2.1 in particular) and the references therein for a more detailed discussion of the history of the leader election problem and its various applications.

The next performance metric that we consider is that of *fault tolerance*. In a distributed computing system with geographically separated processors, it is likely that some processor would go down at some point of time or another. Even the software in the processors may get corrupted and the processors may consequently exhibit malicious

behavior. Communication links between (pairs of) processors can break down as well. All these issues necessitate the study of designing systems that are provably fault-tolerant (for up to a certain number of faults).

The second problem we study in this dissertation is concerned with designing a fault-tolerant system where the *counting* problem — another fundamental problem in distributed computing — can be solved even in the presence of *Byzantine faults*, i.e., where nodes may behave arbitrarily and maliciously. The counting problem, also known as the network size estimation problem, requires that all nodes know the number of nodes in the network, or at the very least, an estimate of it. This knowledge of the network size is required in many other distributed tasks and thus an algorithm to solve the counting problem can be thought of as a basic primitive essential in distributed computing research. Please see Chapter 3 (Section 3.1 in particular) for a more detailed discussion of the counting problem and its countless applications.

The third and the final problem that we study here (see Chapter 4) addresses another fundamental problem in distributed computing, namely building a routable structure in a distributed computing environment. We do this under the presence of a large number of Byzantine (i.e., malicious) nodes and heavy adversarial churn (where nodes leave and new nodes join the network in a very rapid rate). Thus *Byzantine fault tolerance* and *robustness* to heavy dynamism in the network become the relevant performance metrics here.

1.1 The Distributed Computing Model

Let us first formalize how distributed algorithms are executed.

The Structure of Execution of a Distributed Algorithm. It is usually assumed — as we would do throughout this dissertation — that every processor starts with the same piece of code to execute. This does not mean, however, that every processor behaves in identical manner throughout the course of execution of an algorithm. Deviations in behaviors can occur, among other factors, from **(1)** differences in the *initial state* of the processors (e.g., processors may have distinct IDs),[1] and **(2)** *randomization*, where each processor may choose from a *set* of behaviors depending on some randomly generated bit-string.

In describing the behavior of a distributed computing system, it is important to define the notion of a *round* (which will be used later in defining the measure of *time complexity* as well). We would follow the convention where a round consists of the following three steps (for each individual processor P_i):

1. Receive: P_i receives zero or more messages from its neighbors in the network.

2. Compute: Depending on the messages received in the previous step and depending on its current state, P_i does some local computation of its own and then reaches a new state, which may or may not be different from its previous state.

3. Send: Depending on the results of the previous step, P_i sends zero or more messages to zero or more of its neighbors. Those messages would be received by the corresponding recipient processors at the beginning of the next round.

Note that in the point-to-point, message passing communication model,[2] a processor

[1] We note that having distinct IDs does not *necessarily* imply a deviation in processor behaviors.

[2] This is the model we shall be following throughout the dissertation. Please see Section 1.1.1 for more details about the model.

can send messages to or receive messages from its immediate neighbors only.

Remark 1. In an *asynchronous* distributed system, the reception of one or more messages may serve as the "trigger" that initiates the beginning of a new round. But in a synchronous system (which we consider in this dissertation), a new round starts at the same time for every processor; this is made possible thanks to the fact (assumption) that the processors share access to a common, global clock.

Local Knowledge. We assume that nodes initially, i.e., at the beginning of computation, have only limited knowledge: each node knows only about itself and its incident edges; it may not know anything about the identities of its neighbours or their internal states.

Sometimes — but not always — we will assume that nodes have some limited global knowledge, e.g., about the network size (the number of nodes in the network). Such assumptions will always be stated explicitly.

1.1.1 Classification of Distributed Systems

A distributed computing system can be modeled in several different ways, depending on the application domain and on the concerned researcher's priorities. Below we discuss briefly the main parameters that characterize our model of a distributed system.

Shared Memory vs. Message Passing. In the shared memory model, different processors are assumed to have access to a common, shared memory, through which they may (indirectly) communicate with each other.

In contrast, the message passing model addresses the issue of inter-processor communication more explicitly. Here whenever a processor wants to communicate with another

processor, it needs to send a *message* to that processor.

In this dissertation, we shall work with the message passing model exclusively.

Point-to-point Communication Systems vs. Broadcast Networks. There may be variations — even within message-passing systems — depending on the exact mechanism of message passing. For example, in broadcast networks such as Ethernet and radio-based networks (see [105, Section 1.2]), whenever one processor sends out a message, that message may be delivered to multiple other processors simultaneously. In contrast, in point-to-point communication models, a message transfer occurs between specific pairs of processors, i.e., the receiving processor is also unique in this model.

Point-to-point communication models are commondly modeled as an undirected graph where the nodes of the graph are the processors and the edges of the graphs are the communication links between pairs of processors. Note that this graph need not be complete, since every pair of processors may or may not have a communication link between them. We shall usually denote such an undirected graph by $G = (V, E)$, with V denoting the set of nodes and E denoting the set of edges of the graph G. Furthermore, we shall frequently use the symbols n and m to denote the sizes of the node set and the edge set respectively, i.e., $n \stackrel{\text{def}}{=} |V|$ and $m \stackrel{\text{def}}{=} |E|$.

Synchronous vs. Asynchronous. In the synchronous model, the processors are assumed to have access to a common, global clock, which enables the processors to work in lockstep.[3] Thus the execution of any algorithm proceeds in synchronous rounds. In

[3]This does not necessarily imply that different processors have the exact same computing speed; this only states that the different processors begin and finish each round simultaneously (which is made possible

contrast, the asynchronous model makes no assumptions about the synchrony between the processors, and is therefore, more realistic. However, the synchronous model enables us to abstract out the more important features of a distributed system, namely locality and communication costs. As Professor David Peleg says in his book [105],

> "Algorithms for synchronous networks are easier to design, debug, and test than similar algorithms for asynchronous networks. The behavior of asynchronous systems is typically harder to grasp and analyze."

In this dissertation, we shall work with the synchronous model exclusively. We note that this is *not* necessarily an impractical or an unrealistic approach, since there exist mechanisms, called *synchronizers*, that enable one to run any synchronous algorithm on any asynchronous network with only a quadratic slump in the time (round) complexity of the algorithm.

Starting Time — Simultaneous vs. Non-simultaneous Wake-up. We shall usually assume that all nodes start executing the given algorithm at round 0 (zero). This is called the *simultaneous wake-up* model. In contrast, the *non-simultaneous wake-up* model allows nodes to begin execution at different times.

Bandwidth — Local vs. Congest. Depending on the size of messages that is allowed, we can classify distributed computing models into two categories, namely the Local model and the Congest model. In the Local model [105], unbounded message sizes are allowed and so is unlimited local computation. While this may seem unrealistic, these

because of the existence of a common, global clock).

assumptions allow the algorithm designer to focus solely on the issue of *locality*, while doing away with the bothersome issues of *congestion* and local computation.

In contrast, the Congest model captures the real-life restriction of *bandwidth* more accurately by not allowing arbitrarily large messages. Here messages are small-sized, usually of size $O(\log n)$, where n is the number of nodes (processors) in the network. We note that $\log n$ bits are necessary to describe the ID of a node in a network consisting of n nodes (assuming nodes have distinct IDs).

In this dissertation, we shall always work with this latter, more realistic Congest model.

1.1.2 Complexity Measures

Time Complexity. In the *synchronous* model, time is measured by the number of clock ticks called *rounds*. When running a distributed algorithm, different nodes might take a different number of rounds to finish. In that case, the maximum time needed over all nodes is taken as the time complexity. We refer to [105, Definition 2.2.1] for a more formal definition of time complexity.

Message Complexity. This measures the total number of messages that are sent between all pairs of nodes during the computation. In the Congest model, each message has small size, i.e., is assumed to be of $O(\log n)$ bits, where n is the total number of nodes in the network. In the Local model, messages of large sizes can be sent and so the number of messages might not give us an accurate picture of the communication cost. In such situations, the bit complexity, i.e., the number of bits transmitted per edge per round

is a better measure.

1.2 Contributions of This Dissertation: Problems, Results, and Roadmap

Here, we formally state the problems considered in this dissertation, provide an overview of our results, and describe the structure of this dissertation.

[Chapter 2] The (Message) Complexity of Leader Election in Diameter-Two Networks. Leader election is a classical and fundamental problem in distributed computing. The leader election problem requires a group of processors in a distributed network to elect a unique leader among themselves, i.e., exactly one processor must output the decision that it is the leader, say, by changing a special *status* component of its state to the value *leader* [87]. All other nodes must change their status component to the value *non-leader*. These nodes need not be aware of the identity of the leader. This *implicit* variant of leader election is quite standard (see [87]), and has been extensively studied (see e.g., [80] and the references therein) and is sufficient in many applications, e.g., for token generation in a token ring environment [83].

In Chapter 2, we focus on studying the message complexity of leader election (both randomized and deterministic) in synchronous distributed networks, in particular, in networks of *diameter two*. We show the following results:

1. **Algorithms:** We show that the message complexity of leader election in diameter-two graphs is $\tilde{O}(n)$, by presenting a randomized (implicit) leader election algorithm

(see Section 2.2), that takes $O(n \log^3 n)$ messages and runs in $O(1)$ rounds *with high probability* (*whp*).[4] This algorithm works even without knowledge of n. While it is easy to design an $O(n \log n)$ messages randomized algorithm with knowledge of n (see Section 2.1.2), not having knowledge of n makes the analysis more involved.

We also present a *deterministic* algorithm that uses only $O(n \log n)$ messages, but takes $O(\log n)$ rounds. This algorithm needs knowledge of n as well (or at least the knowledge of a constant factor upper bound of $\log n$) (see Section 2.4).

We note that all our algorithms will work seamlessly for complete networks as well.

2. **Lower Bounds:** We show that, in general, it is not possible to improve over our algorithm substantially, by presenting a lower bound for leader election that applies also to randomized algorithms. We show that $\Omega(n)$ messages are needed for any leader election algorithm (regardless of the number of rounds) in a diameter-two network which succeeds with any constant probability that is strictly larger than $\frac{1}{2}$ (see Section 2.3). This lower bound holds even in the Local model [105], where there is no restriction on the number of bits that can be sent on each edge in each round. To the best of our knowledge, this is the first non-trivial lower bound for randomized leader election in diameter-two networks.

   We also present a simple deterministic reduction that shows that any super-linear message lower bound for complete networks also applies to diameter-two networks as well (see Section 2.5). It can be shown that $\Omega(n \log n)$ messages is a lower bound

[4] Throughout, "with high probability" or "whp" is used to mean "with probability at least $1 - n^{-c}$", for some fixed positive constant c.

for deterministic leader election in complete networks [2] (under the assumption that the number of rounds is bounded by some function of n). [5]

By our reduction this lower bound also applies to diameter-two networks.[6]

[Chapter 3] Network Size Estimation in Small-world Networks under Byzantine Faults. In this chapter, we study the fundamental problem of counting the number of nodes in a sparse network (of unknown size) in the presence of a large number of Byzantine nodes. We assume the full information model where the Byzantine nodes have complete knowledge about the entire state of the network at every round (including random choices made by all the nodes), have unbounded computational power, and can deviate arbitrarily from the protocol. Essentially all known algorithms for fundamental Byzantine problems (e.g., agreement, leader election, sampling) studied in the literature assume the knowledge (or at least an estimate) of the size of the network. It is non-trivial to design algorithms for Byzantine problems that work without knowledge of the network size, especially in bounded-degree (expander) networks where the local views of all nodes are (essentially) the same and limited, and Byzantine nodes can quite easily fake the presence/absence of non-existing nodes. To design truly local algorithms that do not rely on any global knowledge (including network size), estimating the size of the network under Byzantine nodes is an important first step.

[5] Afek and Gafni[2] show the $\Omega(n \log n)$ message lower bound for complete networks under the non-simultaneous wakeup model in synchronous networks. As we have verified in our private communications with Shay Kutten (Professor, Technion), the same message bound can be shown to hold in the simultaneous wake-up model as well under the restriction that the number of rounds is bounded by a function of n.

[6] We point out that lower bounds for complete networks do not directly translate to diameter-two networks.

Our main contribution is a randomized distributed algorithm that estimates the size of a network in the presence of a large number of Byzantine nodes. In particular, our algorithm estimates the size of a sparse, "small-world" network[7] with up to $O(n^{1-\delta})$ Byzantine nodes, where n is the (unknown) network size and δ can be any arbitrarily small (but fixed) positive constant. Our algorithm outputs a (fixed) constant factor estimate of $\log n$ with high probability; the correct estimate of the network size will be known to a large fraction ($(1-\varepsilon)$-fraction, for any fixed positive constant ε) of the honest nodes. Our algorithm is fully distributed, lightweight, and simple to implement, runs in $O(\log^3 n)$ rounds, and requires nodes to send and receive only small-sized messages[8] per round; any node's local computation cost per round is also small.

[Chapter 4] Secure, Robust, and Efficient Distributed Computation in Dynamic Peer-to-Peer Networks. In this chapter, we study distributed protocols for constructing and maintaining dynamic network topologies that guarantee efficient and reliable communication even in the presence of a large number of Byzantine (bad) nodes and a continual heavy adversarial churn (i.e., nodes joining and leaving the network continually over time). We assume that the Byzantine nodes have unbounded computational power, can behave arbitrarily and maliciously, and may collude among themselves. However, they are oblivious to the communication between good nodes (i.e., we assume private channels, but no other cryptographic assumptions). We also assume that an adversary controls the churn — it has complete knowledge and control of what nodes join and leave

[7] A "small-world" network is a network with *both* **(1)** high expansion and **(2)** high clustering coefficient. See Section 3.2.1 for the exact definition of the network model.

[8] A "small-sized message" is one that contains a constant number of IDs and $O(\log n)$ additional bits.

and at what time and has unlimited computational power (but is oblivious to the topology changes from round to round).

Our main contribution is a randomized distributed protocol that guarantees with high probability the construction and maintenance of a sparse (logarithmic degree) distributed hash table (DHT) network that guarantees efficient, robust, and reliable communication between (almost all pairs of) good nodes even in the presence of $\frac{n}{\text{polylog}(n)}$ number of Byzantine nodes and *continual heavy* churn of up to $\frac{n}{\text{polylog}(n)}$ *adversarially* chosen nodes joining and leaving every round (where n is the stable network size). Our protocol is efficient, lightweight, and scalable, and it incurs only $\text{polylog}(n)$ overhead for topology construction and maintenance: only polylogarithmic (in n) bits need to be processed and sent by each node per round and any node's computation cost per round is also polylogarithmic in n. To our knowledge, this is the first protocol that can guarantee the above properties under such a large number of Byzantine nodes in a highly dynamic setting.

[Chapter 5] Conclusion. Finally we summarize the main contributions of this dissertation and discuss some interesting problems for further study.

Chapter 2

Message Complexity of Implicit Leader Election in Diameter-two Networks

This chapter focuses on studying the message complexity of implicit leader election in synchronous distributed networks of diameter two[1]. Kutten et al. [80] showed a fundamental lower bound of $\Omega(m)$ (m is the number of edges in the network) on the message complexity of (implicit) leader election that applied also to Monte Carlo[2] randomized algorithms with constant success probability; this lower bound applies for graphs that have diameter at least three. On the other hand, for complete graphs (i.e., graphs with diameter one), Kutten et al. [81] established a tight bound of $\tilde{\Theta}(\sqrt{n})$ on the message

[1]This chapter is based on joint work with Gopal Pandurangan and Peter Robinson; it contains material from [28] and [27].

[2]Recall that a Monte Carlo randomized algorithm is one that may sometimes produce an incorrect solution. In contrast, Las Vegas algorithms are randomized algorithms that *always* produce the correct solution. Please refer to [95, Section 1.2] for a detailed discussion on these two classes of randomized algorithms.

complexity of randomized leader election (n is the number of nodes in the network). For graphs of diameter two, the complexity was not known.

In this chapter, we settle this complexity by showing a tight bound of $\tilde{\Theta}(n)$ on the message complexity of leader election in diameter-two networks. We first give a simple randomized Monte-Carlo leader election algorithm that with high probability (i.e., probability at least $1 - n^{-c}$, for some fixed positive constant c) succeeds and uses $O(n \log^3 n)$ messages and runs in $O(1)$ rounds; this algorithm works without knowledge of n (and hence needs no global knowledge). We then show that any algorithm (even Monte Carlo randomized algorithms with large enough constant success probability) needs $\Omega(n)$ messages (even when n is known), regardless of the number of rounds. We also present an $O(n \log n)$ message deterministic algorithm that takes $O(\log n)$ rounds (but needs knowledge of n); we show that this message complexity is tight for deterministic algorithms.

Together with the two previous results of Kutten et al. [80, 81], our results fully characterize the message complexity of leader election vis-à-vis the graph diameter.

2.1 Introduction

Leader election is a classical and fundamental problem in distributed computing. The leader election problem requires a group of processors in a distributed network to elect a unique leader among themselves, i.e., exactly one processor must output the decision that it is the leader, say, by changing a special *status* component of its state to the value *leader* [87]. All other nodes must change their status component to the value *non-leader*. These nodes need not be aware of the identity of the leader. This *implicit* variant of leader election is quite standard (see e.g., [87]), and has been extensively studied (see

e.g., [80] and the references therein) and is sufficient in many applications, e.g., for token generation in a token ring environment [83].

In another variant, called *explicit* leader election, all the non-leaders change their status component to the value *non-leader*, and moreover, every node must also know the identity of the unique leader. In this variant, $\Omega(n)$ messages is an obvious lower bound (throughout, n denotes the number of nodes in the network) since every node must be informed of the leader's identity. Clearly, any lower bound for implicit leader election applies to explicit leader election as well.

In this chapter, we focus on the implicit variant.

The complexity of leader election, in particular, its message and time complexity, has been extensively studied both in general graphs as well as in special graph classes such as rings and complete networks, see e.g., [87, 106, 110, 120, 81, 80]. While much of the earlier work focused on deterministic algorithms, recent works have studied randomized algorithms (see e.g., [81, 80] and the references therein). Kutten et al. [80] showed a fundamental lower bound of $\Omega(m)$ (m is the number of edges in the network) on the message complexity of (implicit) leader election that applied even to Monte Carlo randomized algorithms with (large-enough) constant success probability; this lower bound applies for graphs that have *diameter at least three*. They also showed that this bound is tight.

On the other hand, for complete graphs (i.e., graphs of diameter one), Kutten et al. [81] established a tight bound of $\tilde{\Theta}(\sqrt{n})$ on the message complexity of randomized leader election (n is the number of nodes in the network).

For graphs of diameter two, the message complexity was not known. In this chapter, we settle this complexity by showing a tight bound of $\tilde{\Theta}(n)$ on the message complexity of

leader election in diameter-two networks. Together with the previous results [80, 81], our results fully characterize the message complexity of leader election vis-à-vis the graph diameter (see Table 2.1).

2.1.1 Our Results

This chapter focuses on studying the message complexity of leader election (both randomized and deterministic) in synchronous distributed networks, in particular, in networks of *diameter two*.

For our algorithms, we assume that the communication is *synchronous* and follows the standard Congest model [105], where a node can send in each round at most one message of size $O(\log n)$ bits on a single edge. We assume that the nodes have unique IDs. We assume that all nodes wake up simultaneously at the beginning of the execution. (Additional details on our distributed computation model are given in Section 2.1.3.)

We show the following results:

1. **Algorithms:** We show that the message complexity of leader election in diameter-two graphs is $\tilde{O}(n)$, by presenting a randomized (implicit) leader election algorithm (see Section 2.2), that takes $O(n \log^3 n)$ messages and runs in $O(1)$ rounds with high probability (whp). This algorithm works even without knowledge of n. While it is easy to design an $O(n \log n)$ messages randomized algorithm with knowledge of n (see Section 2.1.2), not having knowledge of n makes the analysis more involved.

   We also present a *deterministic* algorithm that uses only $O(n \log n)$ messages, but takes $O(\log n)$ rounds. Also this algorithm needs knowledge of n (or at least a constant factor upper bound of $\log n$) (see Section 2.4).

Table 2.1: Message and time complexity of (implicit) leader election

	RANDOMIZED		DETERMINISTIC	
Diameter	Time	Messages	Time	Messages
$D = 1$: [81, 2]				
Upper Bound	$O(1)$	$O(\sqrt{n}\log^{\frac{3}{2}} n)$	$O(1)^{\dagger}$	$O(n\log n)$ †
Lower Bound	$\Omega(1)$	$\Omega(\sqrt{n})$	$\Omega(1)$	$\Omega(n\log n)$
$D \geq 3$: [80]				
Upper Bound	$O(D)$	$O(m\log\log n)$	$O(D\log n)$	$O(m\log n)$
Lower Bound	$\Omega(D)$	$\Omega(m)$	$\Omega(D)$	$\Omega(m)$
$D = 2$:		**Our Results**		
Upper Bound	$O(1)$	$O(n\log^3 n)$	$O(\log n)^{\dagger\dagger}$	$O(n\log n)^{\$}$
Lower Bound	$\Omega(1)$	$\Omega(n)$	$\Omega(1)$	$\Omega(n\log n)$

† Note that attaining $O(1)$ time requires $\Omega(n^{1+\Omega(1)})$ messages in cliques, whereas achieving $O(n\log n)$ messages requires $\Omega(\log n)$ rounds; see [2].

$^{\$}$ Needs knowledge of n.

†† Note that it is easy to give an $O(1)$ round deterministic algorithm that takes $O(m)$ messages.

We note that all our algorithms will work seamlessly for complete networks as well.

2. **Lower Bounds:** We show that, in general, it is not possible to improve over our algorithm substantially, by presenting a lower bound for leader election that applies also to randomized algorithms. We show that $\Omega(n)$ messages are needed for any leader election algorithm (regardless of the number of rounds) in a diameter-two network which succeeds with any constant probability that is strictly larger than $\frac{1}{2}$ (see Section 2.3). This lower bound holds even in the Local model [105], where there is no restriction on the number of bits that can be sent on each edge in each round. To the best of our knowledge, this is the first non-trivial lower bound for randomized leader election in diameter-two networks.

   We also show a simple deterministic reduction that shows that any super-linear message lower bound for complete networks also applies to diameter-two networks as well (see Section 2.5). It can be shown that $\Omega(n \log n)$ messages is a lower bound for deterministic leader election in complete networks [2] (under the assumption that the number of rounds is bounded by some function of n). [3]

   By our reduction this lower bound also applies to diameter-two networks.[4]

[3] Afek and Gafni[2] show the $\Omega(n \log n)$ message lower bound for complete networks under the non-simultaneous wakeup model in synchronous networks. As we have verified in our private communications with Shay Kutten (Professor, Technion), the same message bound can be shown to hold in the simultaneous wake-up model as well under the restriction that the number of rounds is bounded by a function of n.

[4] We point out that lower bounds for complete networks do not directly translate to diameter-two networks.

2.1.2 Technical Overview

All our algorithms exploit the following simple "neighborhood intersection" property of diameter-two graphs: Any two nodes (that are non-neighbors) have at least one neighbor in common (please refer to Observation 1).

Unlike complete networks (which have been extensively studied with respect to leader election — see Section 2.1.5), in diameter-two networks, nodes generally do not have knowledge of n, the network size (in a complete graph, this is trivially known by the degree). This complicates obtaining sublinear in m (where m is the number of edges) message algorithms[5] that are fully localized (don't have knowledge of n).

Indeed, if n is known, the following is a simple randomized algorithm: each node becomes a candidate with probability $\Theta(\frac{\log n}{n})$ and sends its ID to all its neighbors; any node that gets one or more messages acts as a "referee" and notifies the candidate that has the smallest ID (among those it has received). The neighborhood intersection property implies that at least one candidate will be chosen uniquely as the leader with high probability.

If n is not known, the above idea does not work. However, we show that if each node v becomes a candidate with probability $\frac{1+\log(d_v)}{d_v}$, (where d_v is the degree of v) then the above idea can be made to work. The main technical difficulty is then showing that at least one candidate is present (see Section 2.2.1) and in bounding the message complexity (see Section 2.2.2). We use Lagrangian optimization to prove that on expectation at least

[5]Note that $o(m)$ message complexity is achievable only for relatively "dense" graphs, i.e., for graphs where $m = \omega(n)$. As our lower bound (see Theorem 3) shows, $\Omega(n)$ messages are *needed* for any Monte Carlo randomized algorithm with a large enough success probability. Thus for relatively "sparse" graphs, where $m = \Theta(n)$, $\Omega(n) = \Omega(m)$ messages are *necessary* for performing a valid leader election.

$\Theta(\log n)$ candidates will be selected and then use a Chernoff bound to show a high probability result.

Our $\Omega(n)$ randomized lower bound is inspired by the *bridge crossing* argument of [80] and [100]. In [80], the authors construct a "dumbbell" graph G which is done by taking two identical regular graphs G_1 and G_2, removing an edge from each and adding them as bridge edges between G_1 and G_2 (so that regularity is preserved). The argument is that any leader election algorithm should send at least one message across one of the two bridge edges (bridge crossing); otherwise, it can be shown that the executions in G_1 and G_2 are identical leading to the election of two leaders, thus rendering the algorithm invalid. The argument in [80] shows that $\Omega(m)$ messages are needed for bridge crossing. As pointed out earlier in Section 4.1, this construction makes the diameter of G at least three and hence does not work for diameter-two graphs. To overcome this, we modify the construction that takes two complete graphs and add a set of bridge edges (as opposed to just two); see Figure 2.1. This creates a diameter-two graph; however, the large number of bridge edges requires a different style of argument and results in a bound different compared to [80]. We show that $\Omega(n)$ messages (in expectation) are needed to send a message across at least one bridge.

We also present a *deterministic* algorithm that requires $O(n \log n)$ messages, but takes $O(\log n)$ rounds. Note that, in a sense, this improves over the randomized algorithm that sends $O(n \log^3 n)$ messages (although, we did not strive to optimize the log factors). However, the deterministic algorithm is slower by a $\log(n)$-factor and is more involved compared to the very simple randomized algorithm. Our deterministic algorithm uses ideas similar to Afek and Gafni's [2] leader election algorithm for complete graphs;

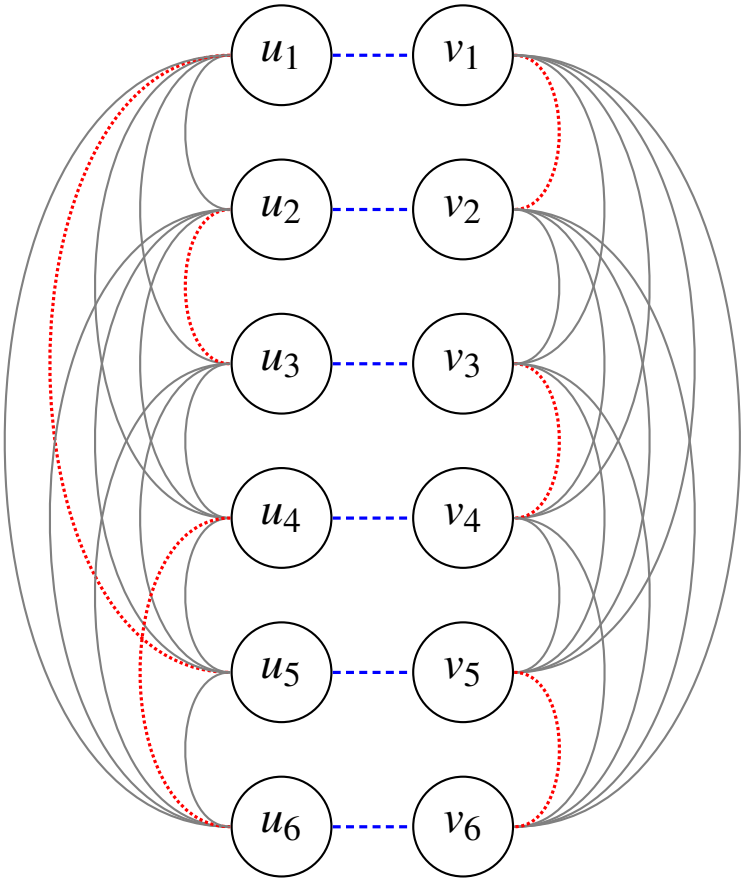

Figure 2.1: The graph used to show the lower bound in Section 2.3

however, the algorithm is a bit more involved. Our algorithm assumes knowledge of n (this is trivially true in complete networks, since every node can infer n from its degree) which is needed for termination. It is not clear if one can design an $O(n \log n)$ messages algorithm (running in say $O(\log n)$ rounds) that does *not* need knowledge of n, which is an interesting open question (see Section 2.6).

Finally, we present a simple reduction that shows that superlinear (in n) lower bounds in complete networks also imply lower bounds for diameter-two networks, by showing how using only $O(n)$ messages and $O(1)$ rounds, a complete network can be converted to a diameter-two network in a distributed manner. This shows that our deterministic algorithm (see Section 2.4) is message optimal.

2.1.3 Distributed Computing Model

The model we consider is similar to the models of [2, 59, 73, 75, 74], with the main addition of giving processors access to a private unbiased coin. We consider a system of n nodes, represented as an undirected graph $G = (V, E)$. In this chapter, we focus on graphs with diameter $D(G) = 2$, where $D(G)$ is the diameter of $G = (V, E)$. An obvious consequence of this is that G is connected, therefore $m \geq n - 1$, where $m = |E|$ and $n = |V|$. Also, since G is not a complete graph, $m < \frac{n(n-1)}{2}$.

Each node has a unique identifier (ID) of $O(\log n)$ bits and runs an instance of a distributed algorithm. The computation advances in synchronous rounds where, in every round, nodes can send messages, receive messages that were sent in the same round by neighbors in G, and perform some local computation. Every node has access to the outcome of unbiased private coin flips (for randomized algorithms). Messages are the

only means of communication; in particular, nodes cannot access the coin flips of other nodes, and do not share any memory. Throughout this chapter, we assume that all nodes are awake initially and simultaneously start executing the algorithm. We note that initially nodes have knowledge only of themselves, in other words we assume the *clean network model* — also called the *KT0 model* [105] which is standard and most commonly used.[6]

2.1.4 Leader Election: Problem Definition

We formally define the leader election problem here.

Every node u has a special variable $\texttt{status}_u$ that it can set to a value in

$$\{\perp, \text{NON-ELECTED}, \text{ELECTED}\};$$

initially we assume $\texttt{status}_u = \perp$.

An *algorithm A solves leader election in T rounds* if, from round T onwards, exactly one node has its status set to ELECTED while all other nodes are in state NON-ELECTED. This is the requirement for standard (implicit) leader election. For *explicit* leader election, we further require that all non-leader nodes should know the identity of the leader.

2.1.5 Other Related Works

The complexity of the leader election problem and algorithms for it, especially deterministic algorithms (guaranteed to always succeed), have been well-studied. Various algorithms and lower bounds are known in different models with synchronous (as well

[6] If one assumes the *KT1 model*, where nodes have an initial knowledge of the IDs of their neighbors, there exists a trivial algorithm for leader election in a diameter-two graph that uses only $O(n)$ messages.

as asynchronous) communication and in networks of varying topologies such as a cycle, a complete graph, or some arbitrary topology (e.g., see [65, 87, 106, 110, 120, 81, 80] and the references therein).

The study of leader election algorithms is usually concerned with both message and time complexity. We discuss two sets of results, one for complete graphs and the other for general graphs. As mentioned earlier, for complete graphs, Kutten et al. [81] showed that $\tilde{\Theta}(\sqrt{n})$ is the tight message complexity bound for randomized (implicit) leader election. In particular, they presented an $O(\sqrt{n \log^3 n})$ messages algorithm that ran in $O(1)$ rounds; they also showed an almost matching lower bound for randomized leader election, showing that $\Omega(\sqrt{n})$ messages are needed for any leader election algorithm that succeeds with a sufficiently large constant probability.

For deterministic algorithms on complete graphs, it is known that $\Theta(n \log n)$ is a tight bound on the message complexity [2]. In particular, Afek and Gafni [2] presented an $O(n \log n)$ messages algorithm for complete graphs that runs in $O(\log n)$ rounds. For complete graphs, Korach et al. [76] and Humblet [59] also presented $O(n \log n)$ message algorithms. Afek and Gafni [2] presented asynchronous and synchronous algorithms, as well as a tradeoff between the message and the time complexity of synchronous *deterministic* algorithms for complete graphs: the results varied from a $O(1)$-time, $O(n^2)$-messages algorithm to a $O(\log n)$-time, $O(n \log n)$-messages algorithm. Afek and Gafni [2], as well as [76, 74] showed a lower bound of $\Omega(n \log n)$ messages for *deterministic* algorithms in the general case. [7]

[7] This lower bound assumes non-simultaneous wakeup though. If nodes are assured to wake up at the same time in synchronous complete networks, there exists a trivial algorithm: if a node's identity is some i, it waits i time before it sends any message then leader election could be solved (deterministically) in $O(n)$

For general graphs, the best known bounds are as follows. Kutten et al. [80] showed that $\Omega(m)$ is a very general lower bound on the number of messages and $\Omega(D)$ is a lower bound on the number of rounds for any leader election algorithm. It is important to point out that their lower bounds apply to graphs with *diameter at least three*. Note that these lower bounds hold even for randomized Monte Carlo algorithms that succeed even with (some large enough, but) constant success probability and apply even to implicit leader election. Earlier results showed such lower bounds only for deterministic algorithms and only for the restricted case of comparison algorithms, where it was also required that nodes may not wake up spontaneously and that D and n were not known. The $\Omega(m)$ and $\Omega(D)$ lower bounds are *universal* in the sense that they hold for all universal algorithms (namely, algorithms that work for all graphs), apply to every $D \geq 3$, m, and n, and hold even if D, m, and n are known, all the nodes wake up simultaneously, and the algorithms can make any use of nodes' identities. To show that these bounds are tight, they presented a randomized algorithm that runs in $O(D)$ rounds (where D is the network diameter) and communicated $O(m \log \log n)$ messages — thus giving an algorithm that is simultaneously almost optimal with respect to both messages and time. They also present a deterministic leader election algorithm — that takes $O(D \log n)$ rounds and exchanges $O(m \log n)$ messages — for general graphs.

Algorithm 1 Randomized leader election in $O(1)$ rounds and $O(n \log^3 n)$ message complexity

1: Each node $v \in V$ selects itself to be a "candidate" with probability $\frac{1+\log(d_v)}{d_v}$, where d_v is the degree of v.

2: **if** v becomes a candidate **then** v sends its ID to all its neighbors.

3: Each node acts as a "referee node" for all its candidate neighbors (including, possibly itself).

4: If a node w receives ID's from its neighbors $v_1, v_2, \ldots, v_j$ (say), then w computes the minimum ID of those and sends it back to those neighbors. That is, w sends $\min\{ID(v_1), ID(v_2), \ldots, ID(v_j)\}$ back to each of $v_1, v_2, \ldots, v_j$.

5: A node v decides that it is the leader if and only if it receives its own ID from *all* its neighbors. Otherwise v decides that it is not the leader.

2.2 A Randomized Algorithm

In this section, we present a simple randomized Monte Carlo algorithm that works in a constant number of rounds. Algorithm 1 is entirely local, as nodes do not require any knowledge of n. Nevertheless, we show that we can sub-sample a small number of candidates (using only local knowledge) that then attempt to become leader. In the remainder of this section, we prove the following result.

Theorem 1. *There exists a Monte Carlo randomized leader election algorithm that, with high probability, succeeds in n-node networks of diameter at most two in $O(1)$ rounds,*

messages on complete graphs in synchronous networks. As we have verified in our private communications with Shay Kutten (Professor, Technion), the same message bound can be shown to hold in the simultaneous wake-up model as well under the restriction that the number of rounds is bounded by a function of n.

while sending $O(n\log^3 n)$ messages.

2.2.1 Proof of Correctness

We use the following property of diameter-two graphs crucially in our algorithm.

Observation 1. Let $G = (V,E)$ be a graph of diameter two. Then for any $u, v \in V$, either $(u,v) \in E$ or $\exists w \in V$ such that $(u,w) \in E$ and $(v,w) \in E$, i.e., u and v have at least one common neighbor w (say).

We note that if one or more candidates are selected, then only the candidate node with the minimum ID is selected as the leader. That is, the leader is unique, and therefore the algorithm produces the correct output. The only case when the algorithm may be wrong is if no candidates are selected to begin with, in which case no leader is selected. In this section, we show that, with high probability, at least two candidates are selected. We note that having only one candidate is sufficient for our purposes, so having two (guaranteed by Lemma 3) or more candidates is actually even better, informally speaking.

We make use of the following fact in order to show that.

Lemma 1. *Let $f(x_1,x_2,\ldots,x_n)$ be a function of n variables $x_1,x_2,\ldots,x_n$, where $x_1,x_2,\ldots,x_n$ are positive reals. f is defined as*

$$f(x_1,x_2,\ldots,x_n) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{1+\log(x_i)}{x_i}.$$

Let C be a real number $\geq n\sqrt{2}$. Then $f(x_1,x_2,\ldots,x_n)$ is minimized, subject to the constraint $\sum_{i=1}^{n} x_i = C$, when $x_i = \frac{C}{n}$, for all $1 \leq i \leq n$. The minimum value that $f(x_1,x_2,\ldots,x_n)$ takes is at the point

$$(\tfrac{C}{n}, \tfrac{C}{n}, \ldots, \tfrac{C}{n}), \text{ and is given by}$$

$$f^{min} = f(\frac{C}{n}, \frac{C}{n}, \ldots, \frac{C}{n}) = \frac{n^2}{C}(1 + \log(\frac{C}{n})).$$

Proof. We use standard Lagrangian optimization techniques to show this. Please refer to the appendix (see Section 2.A) for the full proof. □

Lemma 2. *Let X be a random variable that denotes the total number of candidates selected in Algorithm 1. Then for any $n \geq 119$, the expected number of selected candidates is lower-bounded by*

$$E[X] > 2 + \tfrac{1}{2}\log n.$$

Proof. Let X_v be an indicator random variable that takes the value 1 if and only if v becomes a candidate. Then $\mathrm{E}[X_v] = \Pr[X_v = 1] = \frac{1+\log(d_v)}{d_v}$. Thus if X denotes the total number of candidates selected, then

$$\mathrm{E}[X] = \sum_{v \in V} \mathrm{E}[X_v] = \sum_{v \in V} \frac{1 + \log(d_v)}{d_v}.$$

Since G is connected, $m \geq n-1 \implies 2m \geq 2n-2 > n\sqrt{2}$, i.e., the precondition for the applicability of Lemma 1 is satisfied.

Thus by By Lemma 1, $\mathrm{E}[X]$ is minimized subject to the constraint

$$\sum_{v \in V(G)} d_v = 2m,$$

when $d_v = \frac{2m}{n}$ for all $v \in V(G)$, i.e., when G is regular.

Case 1 ($n-1 \leq m \leq n^{\frac{3}{2}}$)**:** The minimum value that $\mathrm{E}[X]$ takes is given by

$$\begin{aligned}
\mathrm{E}[X]|_{\min} &= \frac{n^2}{2m}(1+\log(\frac{2m}{n})) \\
&> \frac{n^2}{2m} && \text{(since } 1+\log(\tfrac{2m}{n}) > 1\text{)} \\
&\geq \frac{\sqrt{n}}{2} && \text{(since } m \leq n^{\frac{3}{2}}\text{)}
\end{aligned}$$

Case 2 ($n^{\frac{3}{2}} < m \leq \binom{n}{2}$)**:** The minimum value that $\mathrm{E}[X]$ takes is given by

$$\begin{aligned}
\mathrm{E}[X]|_{\min} &= \frac{n^2}{2m}(1+\log(\frac{2m}{n})) \\
&> 1+\log(\frac{2n^{\frac{3}{2}}}{n}) && \text{(since } \tfrac{n^2}{2m} > 1 \text{ and } m > n^{\frac{3}{2}}\text{)} \\
&= 1+\log 2+\log(n^{\frac{1}{2}}) = 2+\frac{1}{2}\log n.
\end{aligned}$$

Finally, we note that $\frac{\sqrt{n}}{2} > 2+\frac{1}{2}\log n$ for all $n \geq 119$, and this completes the proof.

□

Lemma 3. *If X denotes the number of candidates selected, then* $\Pr[X \leq 1] < n^{-\frac{1}{4}}$.

Proof. We set $\delta = \frac{2+\log n}{4+\log n}$. Then clearly $0 < \delta < 1$, and $1-\delta = \frac{2}{4+\log n}$. Also, substituting the lower bound for μ from Lemma 2, we have that

$$(1-\delta)\mu > 1. \tag{2.1}$$

We want to apply Chernoff bound, so we first compute a lower bound for the quantity

$\frac{\mu\delta^2}{2}$:

$$\begin{aligned}
\frac{\mu\delta^2}{2} &= (\frac{\mu}{2})\cdot\delta^2 \\
&> (\frac{4+\log n}{2})\cdot(\frac{2+\log n}{4+\log n})^2 && \text{(since } \mu > \tfrac{4+\log n}{2} \text{ from Lemma 2)} \\
&= \frac{(2+\log n)^2}{4(4+\log n)} = \frac{4+\log n(4+\log n)}{4(4+\log n)} \\
&= \frac{1}{4+\log n} + \frac{\log n}{4} > \frac{\log n}{4} && (2.2)
\end{aligned}$$

Hence from Equation 2.1, we have that

$$\begin{aligned}
\Pr[X \leq 1] &\leq \Pr[X \leq (1-\delta)\mu] \\
&\leq \exp(-\frac{\mu\delta^2}{2}) && \text{(by Chernoff bound [94, Theorem 4.5(2)])} \\
&< \exp(-\frac{\log n}{4}) && \text{(since } \tfrac{\mu\delta^2}{2} > \tfrac{\log n}{4} \text{ from Equation (2.2))} \\
&< 2^{-\frac{\log n}{4}} = n^{-\frac{1}{4}}.
\end{aligned}$$

□

2.2.2 Computing the Message Complexity

We use the following variant of Chernoff Bound [94, Theorem 4.4(3)] in the following analysis.

Theorem 2 (Chernoff Bound)**.** *Let $X_1, X_2, \ldots, X_n$ be independent indicator random variables, and let $X = \sum_{i=1}^{n} X_i$. Then the following Chernoff bound holds: for $R \geq 6E[X]$,* $\Pr[X \geq R] \leq 2^{-R}$.

Now on to the analysis.

Computing the expected total message complexity of the algorithm. Note that the expected total message complexity of the algorithm can be bounded as follows. Let M^{entire} be a random variable that denotes the total messages sent during the course of the algorithm. Let M_v be the number of messages sent by node v. Thus

$$M^{\text{entire}} = \sum_{v \in V} M_v.$$

A node v becomes a candidate with probability $\frac{1+\log(d_v)}{d_v}$ and, if it does, it sends d_v messages (the referee nodes reply to these, but that increases the total number of messages by only a factor of 2). Hence by linearity of expectation, it follows that

$$\begin{aligned}
\mathrm{E}[M^{\text{entire}}] &= \sum_{v \in V} \mathrm{E}[M_v] = \sum_{v \in V} 2\frac{1+\log(d_v)}{d_v} d_v \\
&= 2\sum_{v \in V} (1+\log(d_v)) \leq 2\sum_{v \in V} (1+\log n) \\
&\leq 2n + 2n\log n.
\end{aligned}$$

Showing concentration by introducing the idea of "buckets". To show concentration, we cannot directly apply a standard Chernoff bound, since that works for 0-1 random variables only, whereas the M_vs are not 0-1 random variables (they take values either 0 or d_v). To handle this, we "bucket" the degrees into different categories, called "buckets", based on their value; we then use a Chernoff bound as detailed below.

Definition 1. *Let k be a positive integer such that $2^{k-1} < n \leq 2^k$. For $0 \leq j \leq k$, let $V_j \subset V$ be the set of vertices with degree in $(2^{j-1}, 2^j]$, i.e., if $v \in V_j$, then $2^{j-1} < d_v \leq 2^j$.*

Thus

$$V_0 \stackrel{\text{def}}{=} \{v \in V \mid d_v = 1\},$$

$$V_1 \stackrel{\text{def}}{=} \{v \in V \mid d_v = 2\},$$

$$V_2 \stackrel{\text{def}}{=} \{v \in V \mid d_v = 3 \text{ or } d_v = 4\},$$

$$V_3 \stackrel{\text{def}}{=} \{v \in V \mid d_v \in \{5,6,7,8\}\}, \text{ and so on.}$$

Remark 2. We note that

$$\sum_{j=0}^{k} n_j = n, \text{ where } n_j = |V_j| \text{ for } 0 \leq j \leq k.$$

In particular, $n_j \leq n$ for all $j \in [0,k]$.

Intuition behind this idea of "bucketing".

- Nodes are bucketed according to the logarithm of their respective degree (ceiling of the logarithm, to be more precise) (see Definition 1). Thus nodes in the same bucket send approximately the same amount of messages.

- Nodes in a higher-degree bucket will send a higher number of messages *each*, than nodes in a lower-degree bucket. The greater effect of a higher-degree bucket, however, is counterbalanced by the the smaller number (in expectation) of candidates in that bucket.

- Thus the number of messages contributed by each bucket — whether high-degree or low-degree — is approximately the same (see Lemma 4).

- We can, therefore, easily "add" the contributions made by each bucket — there are at most $(\lceil \log n \rceil + 1)$ such buckets — and obtain the combined (total) message complexity (see Lemma 5).

Counting the number of messages sent in the first round by each individual node.

1. **Analyzing vertices with degree ≤ 2:** We recall that X_v is an indicator random variable that takes the value 1 if and only if v becomes a candidate. Then $\Pr[X_v = 1] = 1$ if $v \in V_0 \cup V_1$, i.e., every vertex with degree 1 or degree 2 selects itself to be a candidate, deterministically.

   For $v \in V$, let m_v denote the number of messages that v sends. So $m_v = d_v$ if v becomes a candidate, and $m_v = 0$ otherwise. Let M_j be the total number of messages that members of V_j send, i.e.,

$$\begin{aligned} M_j &\stackrel{\text{def}}{=} \sum_{v \in V_j} m_v \leq \sum_{v \in V_j} d_v \\ &\leq \sum_{v \in V_j} 2^j = n_j . 2^j \leq n . 2^j \\ &\implies M_0 \leq n \text{ and } M_1 \leq 2n. \end{aligned}$$

2. **Analyzing vertices with degree > 2:** We recall that for $v \in V$, X_v is an indicator random variable that takes the value 1 if and only if v becomes a candidate. Let i be an integer in $[2, k]$ and let $v \in V_i$. Then

   Observation 2. $\frac{i}{2^i} < \mathrm{E}[X_v] < \frac{3i}{2^i}$.

Proof. For $v \in V_i$, $2^{i-1} < d_v \leq 2^i$. So

$$\begin{aligned}
\mathrm{E}[X_v] &= \Pr[X_v = 1] && \text{(since } X_v \text{ is an indicator random variable)} \\
&= \frac{1+\log(d_v)}{d_v} \\
&\implies \frac{1+\log(2^{i-1})}{2^i} < \mathrm{E}[X_v] < \frac{1+\log(2^i)}{2^{i-1}} && \text{(since } 2^{i-1} < d_v \leq 2^i\text{)} \\
&\text{or, } \frac{i}{2^i} < \mathrm{E}[X_v] < \frac{i+1}{2^{i-1}} \leq \frac{3i}{2^i} && \text{(since } i \geq 2 \implies \tfrac{3i}{2} \geq i+1\text{)}
\end{aligned}$$

□

For $0 \leq j \leq k$, let Y_j be a random variable that denotes the total number of candidates selected from V_j.

Observation 3. For $2 \leq i \leq k$, $\frac{in_i}{2^i} < \mathrm{E}[Y_i] < \frac{3in_i}{2^i}$.

Proof.

$$\begin{aligned}
Y_i = \sum_{v \in V_i} X_v &\implies \mathrm{E}[Y_i] = \mathrm{E}[\sum_{v \in V_i} X_v] \\
&= \sum_{v \in V_i} \mathrm{E}[X_v] && \text{(by linearity of expectation)} \\
&\implies \sum_{v \in V_i} \frac{i}{2^i} < \mathrm{E}[Y_i] < \sum_{v \in V_i} \frac{3i}{2^i} \\
&\implies \frac{in_i}{2^i} < \mathrm{E}[Y_i] < \frac{3in_i}{2^i}.
\end{aligned}$$

□

Remark 3. $\forall u, v \in V(G)$, $u \neq v$, X_u and X_v are independent, and for $0 \leq j \leq k$, we define Y_j as

$$Y_j = \sum_{v \in V_j} X_v,$$

i.e., Y_j is a sum of independent, 0-1 random variables. Hence we can use Theorem 2 to show that Y_j is concentrated around its mean (expectation).

We recall that for $0 \leq j \leq k$, M_j is the total number of messages that members of V_j send, i.e., for $2 \leq i \leq k$,

$$M_i = \sum_{v \in V_i} m_v = \sum_{v \in V_i, X_v = 1} d_v.$$

Lemma 4. *For any integer $i \in [2,k]$, it holds that*

$$\Pr[M_i \geq 24n\log^2 n] \leq \tfrac{1}{n^4}.$$

Proof.

$$\begin{aligned}
M_i = \sum_{v \in V_i} m_v &= \sum_{v \in V_i, X_v = 1} d_v \\
\implies \sum_{v \in V_i, X_v = 1} 2^{i-1} &< M_i \leq \sum_{v \in V_i, X_v = 1} 2^i \qquad \text{(since } 2^{i-1} < d_v \leq 2^i\text{)} \\
\implies 2^{i-1}.Y_i &< M_i \leq 2^i.Y_i.
\end{aligned}$$

Case 1 **($\mathbf{E}[Y_i] = 0$):** $\mathrm{E}[Y_i] = 0$ if and only if $n_i = 0$, i.e., if and only if $\nexists v \in V$ such that $2^{i-1} < d_v \leq 2^i$. But $n_i = 0 \implies V_i = \phi$, the empty set. Therefore, $M_i = 0$.

Case 2 **($0 < \mathbf{E}[Y_i] < 1$):** Assuming $n \geq 3$, $4\log n > 6 > 6\mathrm{E}[Y_i]$. Therefore, by Theorem 2,

$$\begin{aligned}
\Pr[Y_i \geq 4\log n] &\leq 2^{-4\log n} = n^{-4} \\
\implies \Pr[M_i \geq 2^i.4\log n] &\leq n^{-4} \qquad \text{(since } M_i \leq 2^i.Y_i\text{)} \\
\implies \Pr[M_i \geq 8n\log n] &\leq n^{-4} \qquad \text{(since } i \leq k < \log n + 1\text{)}
\end{aligned}$$

Case 3 ($\mathbf{E}[Y_i] \geq 1$)**:** We have shown before that $\mathrm{E}[Y_i] \leq \frac{3in_i}{2^i}$. But $n_i \leq n$ for all $2 \leq i \leq k$. Hence $\mathrm{E}[Y_i] \leq \frac{3ni}{2^i}$. Assuming $n \geq 3$, $4\log n > 6$. Therefore, by Theorem 2,

$$\begin{aligned}
&\Pr[Y_i \geq 12n\log n.\frac{i}{2^i}] \leq \Pr[Y_i \geq 4\log n\mathrm{E}[Y_i]] \\
&\leq 2^{-4\log n\mathrm{E}[Y_i]} = n^{-4\mathrm{E}[Y_i]} \leq n^{-4} && \text{(since } \mathrm{E}[Y_i] \geq 1\text{)} \\
&\implies \Pr[M_i \geq 12in\log n] \leq n^{-4} && \text{(since } M_i \leq 2^i.Y_i\text{)}
\end{aligned}$$

But we have $i \leq k < \log n + 1 < 2\log n \implies 12in\log n < 24n\log^2 n$. Hence

$$\Pr[M_i \geq 24n\log^2 n] \leq \Pr[M_i \geq 12in\log n] \leq n^{-4}.$$

□

Combining the effects of all the nodes.

Lemma 5. *If M denotes the total number of messages sent by the candidates (in the first round only), then*

$$\Pr[M \geq 27n\log^3 n] < \tfrac{1}{n^3}.$$

Proof.

$$\begin{aligned}
M &\stackrel{\text{def}}{=} \sum_{i=0}^{k} M_i = M_0 + M_1 + \sum_{i=2}^{k} M_i \\
&\leq n + 2n + \sum_{i=2}^{k} M_i && \text{(since } M_0 \leq n \text{ and } M_1 \leq 2n\text{)} \\
&= 3n + \sum_{i=2}^{k} M_i.
\end{aligned}$$

But for $2 \leq i \leq k$, $\Pr[M_i \geq 24n\log^2 n] \leq \frac{1}{n^4}$. Taking the union bound over $2 \leq i \leq k$,

$$\Pr[M_{i'} \geq 24n\log^2 n] \text{ for some } i' \in [2,k] \text{ is } \leq \frac{\log n}{n^4} < \frac{1}{n^3}$$

$$\implies \Pr[\sum_{i=2}^{k} M_i \geq 24n\log^3 n] < \frac{1}{n^3}$$

$$\implies \Pr[3n + \sum_{i=2}^{k} M_i \geq 3n + 24n\log^3 n] < \frac{1}{n^3}$$

$$\implies \Pr[M \geq 3n + 24n\log^3 n] < \frac{1}{n^3} \qquad \text{(since } M \leq 3n + \textstyle\sum_{i=2}^{k} M_i\text{)}$$

But $3n \leq 3n\log^3 n$ for $n \geq 2$, so, $3n + 24n\log^3 n \leq 27n\log^3 n$. Hence

$$\Pr[M \geq 27n\log^3 n] \leq \Pr[M \geq 3n + 24n\log^3 n] < \frac{1}{n^3}.$$

□

Lemma 6. *If M^{entire} denotes the total number of messages sent during the entire run of Algorithm 1, then*

$$\Pr[M^{entire} \geq 54n\log^3 n] < \tfrac{1}{n^3}.$$

Proof. Let M' denote the number of messages sent by the "referee" nodes in the *second* round of the algorithm. We recall that M is the number of messages sent by the "candidate" nodes in the *first* round of the algorithm. Then $M' \leq M$, and $M^{\text{entire}} = M + M' \leq 2M$, and the result follows. □

This completes the proof of Theorem 1.

2.3 A Lower Bound for Randomized Algorithms

In this section we show that $\Omega(n)$ is a lower bound on the message complexity for solving leader election with any randomized algorithm in diameter-two networks. Recall that

[81] shows a lower bound of $\Omega(\sqrt{n})$ for the special case of diameter-one networks, and we know from [80] that the message complexity becomes $\Omega(m)$ for (most) diameter-three networks. Thus Theorem 3 completes the picture regarding the message complexity of leader election when considering networks according to their diameter.

Theorem 3. *Any algorithm that solves implicit leader election with probability at least $\frac{1}{2}+\varepsilon$ in any n-node network with diameter at most two, for any constant $\varepsilon > 0$, sends at least $\Omega(n)$ messages in expectation. This holds even if nodes have unique IDs and know the network size n.*

In the remainder of this section, we prove Theorem 3.

Proof by Contradiction. Assume towards a contradiction, that there is an algorithm that elects a leader with probability $\frac{1}{2}+\varepsilon$ that sends $o(n)$ messages with probability approaching 1. In other words, we assume that the event where the algorithm sends at least $\Omega(n)$ messages (of arbitrary size) happens with probability at most $o(1)$.

Unique IDs vs. Anonymous. Before describing our lower bound construction, we briefly recall a simple reduction used in [81] that shows that assuming unique IDs does not change the success probability of the algorithm by more than $\frac{1}{n}$: Since we assume that nodes have knowledge of n, it is straightforward to see that nodes can obtain unique IDs (whp) by choosing a random integer in the range $[1, n^c]$, for some constant $c \geq 4$. Thus, we can simulate an algorithm that requires unique IDs in the anonymous case and the simulation will be correct with high probability. Suppose that there is an algorithm A that can break the message complexity bound of Theorem 3 while succeeding with probability $\geq \frac{1}{2}+\varepsilon$, for some positive constant ε, when nodes have unique IDs. Then, the above

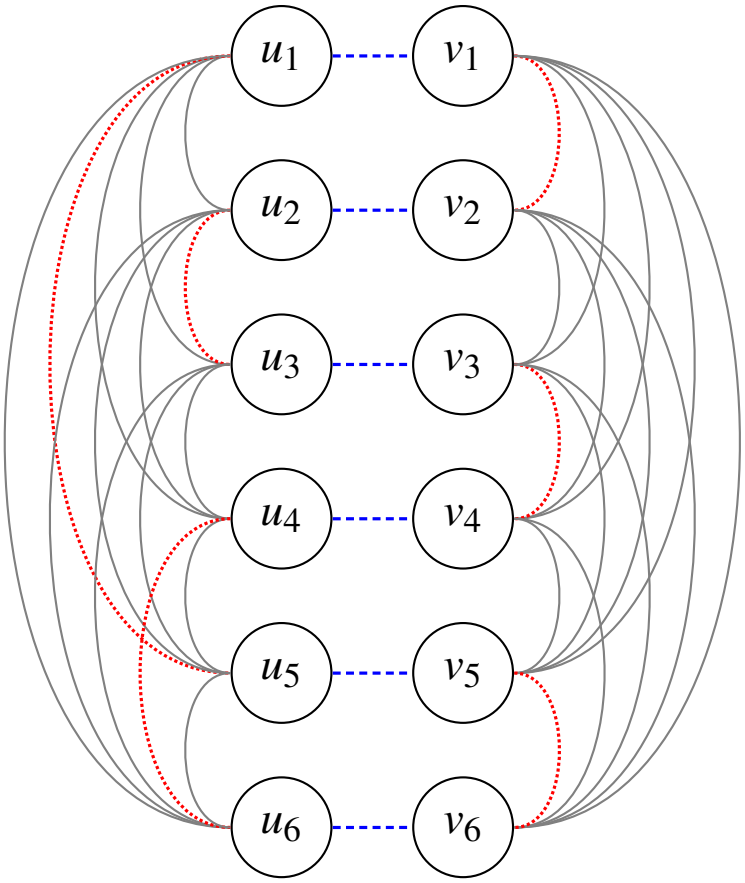

Figure 2.2: The lower bound graph construction used in Theorem 3 for $n = 12$, with cliques C_1 and C_2, where $V(C_1) = \{u_1, \ldots, u_6\}$ and $V(C_2) = \{v_1, \ldots, v_6\}$. The dotted red edges are the edges in M_1 and M_2 that are removed from C_1 and C_2 when constructing G and the blue dashed inter-clique edges are given by the maximal matching M between C_1 and C_2. Each blue edge incident to some node u_i is connected by using the port number of u_i's (removed) red edge.

simulation yields an algorithm A' that works in the case where nodes are *anonymous* with the same message complexity bound as algorithm A and succeeds with probability at least $(\frac{1}{2}+\varepsilon-\frac{1}{n}) \geq \frac{1}{2}+\varepsilon'$, for some positive constant ε'. We conclude that proving the lower bound for the anonymous case is sufficient to imply a lower bound for the case where nodes do have unique IDs.

The Lower Bound Graph. Our lower bound is inspired by the bridge crossing argument of [80] and [100]. For simplicity, we assume that $\frac{n}{4}$ is an integer. Consider two cliques C_1 and C_2 of $\frac{n}{2}$ nodes each and let G' be the n-node graph consisting of the two (disjoint) cliques. The *port numbering* of an edge $e = (u_i, v_j) \in E(G')$ refers to the port number at u_i and the respective port number at v_j that connects e. The port numberings of the edges define an *instance of* G'.

Given an instance of G', we will now describe how to obtain an instance of graph G that has the same node set as G'. Fix some arbitrary enumeration $u_1, \ldots, u_{\frac{n}{2}}$ of the nodes in C_1 and similarly let $v_1, \ldots, v_{\frac{n}{2}}$ be an enumeration of the nodes in C_2.[8] To define the edges of G, we randomly choose a maximal matching M_1 of $\frac{n}{4}$ edges in the subgraph C_1. Consider the set of edges $M_2' = \{(v_i, v_j) \mid \exists (u_i, u_j) \in M_1\}$, which we can think of as a mapping of the matching M_1 to C_2. Then, we define M_2 to be a randomly chosen maximal matching on C_2 restricted to the edges in $E(G') \setminus M_2'$. Next, we remove all edges in $M_1 \cup M_2$ from G'. So far, we have obtained a graph where each node has one *unwired port*.

The edge set of G consists of all the remaining edges of G' in addition to the set of

[8] This enumeration is used only for the description of the lower bound construction and is unbeknownst to the nodes.

bridge edges

$$M = \{(u_1, v_1), \ldots, (u_{\frac{n}{2}}, v_{\frac{n}{2}})\},$$

where we connect these bridge edges by using the unwired ports that we obtained by removing the edges as described above. We say that an edge is an *intra-clique edge* if it has both endpoints in either C_1 or C_2. Observe that the intra-clique edges of G are a subset of the edges of G'. Figure 2.2 gives an illustration of this construction.

Lemma 7. *Graph G is an n-node network of diameter two and the port numbering of each intra-clique edge in G is the same as that of the corresponding edge in G'.*

Proof. By construction, each node in C_1 has the same port numbering in both graphs, except for its (single) incident edge that was replaced with a bridge edge to some node in C_2. Thus we focus on showing that G has diameter two.

We will show that node $u_i \in C_1$ has a path of length two to every other node in G. Observe that any two nodes $u_i, u_j \in C_1$ each have $\frac{n}{2} - 2$ incident intra-clique edges and since $\frac{n}{2} - 2 > \frac{|C_1|}{2}$ they must have a common neighbor. Now, consider some node $v_j \in C_2$ and assume that $j \neq i$, as otherwise there is the bridge edge $(u_i, v_i) \in M \subseteq E(G)$, and we are done. If $(u_i, u_j) \in E(G)$, then the result follows because $(u_j, v_j) \in M$. Thus, assume $(u_i, u_j) \notin E(G)$. It follows that there is a path $u_i \to v_i \to v_j$ in G, since, by construction, the edge $(v_i, v_j) \in M_2'$ and $(v_i, v_j) \notin M_2$, thus $(v_i, v_j) \in E(G)$.

A symmetric argument shows that every node has distance ≤ 2 from a given node in C_2. □

A *state* σ of the nodes in C_1 is a $\frac{n}{2}$-size vector of the local states of the $\frac{n}{2}$ nodes in C_1. Since we assume that nodes are anonymous, a state σ that is reached by the nodes in C_1

can also be reached by the nodes in C_2. More formally, when executing the algorithm on the disconnected network G', we can observe that every possible state σ (of $\frac{n}{2}$ nodes) has the same probability of occurring in C_1 as in C_2. Thus, a state where there is exactly one leader among the $\frac{n}{2}$ nodes of a clique in G', is reached with some specific probability q depending on the algorithm. By a slight abuse of notation, we also use G' and G to denote the event that the algorithm executes on G' respectively G. For the probability of the event One, which occurs when there is exactly 1 leader among the n nodes, we get

$$\Pr\left[\text{One} \;\middle|\; G'\right] = 2q(1-q) \leq \frac{1}{2}, \tag{2.3}$$

which holds for any value of q. Since G' is disconnected, the algorithm does not need to succeed with nonzero probability when being executed on G'. However, below we will use this observation to obtain an upper bound on the probability of obtaining (exactly) one leader in G.

Now consider the execution on the diameter-two network G (obtained by modifying the ports of G' as described above) and let $C_1 \nleftrightarrow C_2$ be the event that no message is sent across the bridges between C_1 and C_2. Since we assume the port numbering model where nodes are unaware of their neighbors initially, it follows by Lemma 7 that

$$\Pr\left[\text{One} \mid C_1 \nleftrightarrow C_2, G\right] = \Pr\left[\text{One} \;\middle|\; G'\right]. \tag{2.4}$$

Let M be the event that the algorithm sends $o(n)$ messages. Recall that we assume towards a contradiction that $\Pr[M \mid G] = 1 - o(1)$.

Lemma 8. $\Pr[C_1 \leftrightarrow C_2 \mid G, M] = o(1)$.

Proof. The proof is inspired by the guessing game approach of [50] and Lemma 16 in [100]. Initially, any node $u \in C_1$ has $\frac{n}{2} - 1$ ports that are all equally likely (i.e., with

probability $\frac{1}{\frac{n}{2}-1}$) to be connected to the (single) bridge edge incident to u. As u sends messages to other nodes, it might learn about some of its ports connecting to non-bridge edges and hence this probability can increase over time. However, we condition on event M, i.e., the algorithm sends at most $o(n)$ messages in total and hence at least $\frac{n}{4}$ ports of each node u remain unused at any point.

It follows that the probability of some node u to send a message over a (previously unused) port that connects a bridge edge is at most $\frac{4}{n}$ at any point of the execution. Let X be the total number of ports connecting bridge edges over which messages are sent during the run of the algorithm and let X_u be the indicator random variable that is 1 iff node u sends a message across its bridge edge. Let S_u be the number of messages sent by node u. It follows by the hypergeometric distribution that

$$\mathrm{E}[X_u \mid G, M] = \frac{S_u}{\Theta(n)}$$

for each node u and hence,

$$\mathrm{E}[X \mid G, M] = \sum_{u \in V(G)} \frac{S_u}{\Theta(n)} = \frac{1}{\Theta(n)} \sum_{u \in V(G)} S_u = o(1),$$

where we have used the fact that $\sum_{u \in V(G)} S_u = o(n)$ due to conditioning on event M. By Markov's Inequality, it follows that the event $C_1 \leftrightarrow C_2$, i.e., $X \geq 1$, occurs with probability at most $o(1)$. □

We now combine the above observations to obtain

$$\begin{aligned}
&\Pr[\text{One} \mid G, M] && (2.5)\\
&= \Pr[\text{One} \mid C_1 \nleftrightarrow C_2, G, M] \Pr[C_1 \nleftrightarrow C_2 \mid G, M] \\
&+ \Pr[\text{One} \mid C_1 \leftrightarrow C_2, G, M] \Pr[C_1 \leftrightarrow C_2 \mid G, M] \\
&\leq \Pr[\text{One} \mid C_1 \nleftrightarrow C_2, G, M] + o(1) && \text{(by Lemma 8)}\\
&\leq \frac{1}{2} + o(1), && (2.6)
\end{aligned}$$

where the last inequality follows by first using (2.4) and noting that the upper bound (2.3) still holds when conditioning on the event M.

Finally, we recall that the algorithm succeeds with probability at least $\frac{1}{2} + \varepsilon$ and $\Pr[M \mid G] \geq 1 - o(1)$, which yields

$$\frac{1}{2} + \varepsilon \leq \Pr[\text{One} \mid G] \leq \Pr[\text{One} \mid G, M] + o(1) \leq \frac{1}{2} + o(1),$$

which is a contradiction, since we have assumed that $\varepsilon > 0$ is a constant.

This completes the proof of Theorem 3.

2.4 A Deterministic Algorithm

Our algorithm (Algorithm 2) is inspired by the solution of Afek and Gafni [2] for the n-node clique. However, there are some complications that we explain below, since we cannot rely on every pair of nodes to be connected by an edge. Note that our algorithm assumes that n (or a constant factor upper bound for $\log n$) is known to all nodes.

For any node $v \in V$, we denote the degree of v by d_v and the ID of v by ID_v. At any time-point in the algorithm, L_v denotes the highest ID that v has so far learned (among

Algorithm 2 Deterministic Leader Election in $O(\log n)$ rounds and with $O(n \log n)$ messages: Code for a node v

1: v becomes a "candidate" and "active".
2: $L_v \leftarrow ID_v$.
3: $N_v \leftarrow ID_v$.
4: v numbers its neighbors from 1 to d_v, which are called $w_{v,1}, w_{v,2}, \ldots, w_{v,d_v}$ respectively.
5: **for** phase $i = 1$ to $\lceil \log n \rceil$ **do** ▷ This *uniformity hypothesis* is needed only for termination.
6: **if** v is active **then**
7: v sends a "probe" message containing its ID to its neighbors $w_{v,2^{i-1}}, \ldots, w_{v,\min\{d_v, 2^i-1\}}$.
8: **if** $d_v \leq 2^i - 1$ **then** ▷ If v is finished with exploring its adjacency list, v becomes inactive.
9: v becomes inactive.
10: **end if**
11: **end if**
12: Let $\mathcal{X}_v$ be the set (possibly empty) of neighbors of v from whom v receives messages in this round.
13: Let $\mathrm{ID}(\mathcal{X}_v)$ be the set of ID's sent to v by the members of $\mathcal{X}_v$.
14: Let ID_u be the highest ID in $\mathrm{ID}(\mathcal{X}_v)$.
15: **if** $ID_u > L_v$ **then**
16: $L_v \leftarrow ID_u$. ▷ v stores the highest ID seen so far in L_v.
17: v tells N_v about $L_v = ID_u$, i.e., the highest ID it has seen so far.
18: $N_v \leftarrow x$. ▷ v remembers neighbor who told v about L_v.
19: v becomes "inactive" and "non-candidate".
20: **end if**
21: v tells every member of $\mathcal{X}_v$ about L_v, i.e., the highest ID it has seen so far.
22: **end for**
23: **if** v is still a candidate **then**
24: v elects itself to be the leader.
25: **end if**

all the probe messages it has received, in the current round or in some previous round).

The algorithm proceeds as a sequence of $\lceil \log n \rceil$ phases. Initially every node is a "candidate" and is "active". Each node v numbers its neighbors from 1 to d_v, denoted by

$$w_{v,1}, w_{v,2}, \ldots, w_{v,d_v}$$

respectively. In phase i, if a node v is active, v sends probe-messages containing its ID to its neighbors $w_{v,2^{i-1}}, \ldots, w_{v,k}$, where $k = \min\left\{d_v, 2^i - 1\right\}$. [9] Each one of them replies back with the highest ID it has seen so far. If any of those ID's is higher than ID_v, then v stops being a candidate and becomes inactive. Node v also becomes inactive if it has finished sending probe-messages to all its neighbors. After finishing the $\lceil \log n \rceil$ phases v becomes the leader if it is still a candidate.

The idea behind the algorithm is to exploit the *neighborhood intersection property* (see Observation 1) of diameter-two networks. Since for any $u, v \in V$, there is an $x \in V$ that is connected to both u and v (unless u and v are directly connected via an edge) and acts as a "referee" node for candidates u and v. This means that x serves to inform u and v who among them is the winner, i.e., has the higher ID. Thus at the end of the algorithm, every node except the one with the highest ID should know that he is not a leader. We present the formal analysis of Theorem 4 in Sections 2.4.1 and 2.4.2.

Theorem 4. *There exists a deterministic leader election algorithm for n-node networks with diameter at most two that sends $O(n \log n)$ messages and terminates in $O(\log n)$ rounds.*

[9] We note that this is the main idea borrowed from the Afek-Gafni algorithm [2] — the number of messages sent by each "active" node *increases* exponentially in every phase. That effect is, however, counterbalanced by the exponentially *decreasing* number of "active" nodes.

In the pseudocode and the subsequent analysis we use v and ID_v interchangeably to denote the node v.

2.4.1 Proof of Correctness

Define $v^{\max}$ to be the node with the highest ID in G.

Lemma 9. *v^{max} becomes a leader.*

Proof. Since $v^{\max}$ has the highest ID in G, the if-clause of Line 15 of Algorithm 2 is never satisfied for $v^{\max}$. Therefore $v^{\max}$ never becomes a non-candidate, and hence becomes a leader at the end of the algorithm. □

Lemma 10. *No other node except v^{max} becomes a leader.*

Proof. Consider any $u \in V$ such that $u \neq v^{\max}$.

- **Case** 1 (**$v^{\max}$ and u are connected via an edge):** Since $v^{\max}$ has the highest ID in G, the if-clause of Line 15 of Algorithm 2 is never satisfied for $v^{\max}$. Therefore $v^{\max}$ becomes inactive only if it has already sent probe-messages to all its neighbors (or $v^{\max}$ never becomes inactive). In particular, u always receives a probe-message from $v^{\max}$ containing $ID_{v^{\max}}$. Since $ID_{v^{\max}} > ID_u$, u becomes a non-candidate at that point (if u was still a candidate until that point) and therefore does not become a leader.

- **Case** 2 (**$v^{\max}$ and u do not have an edge between them):** By Observation 1, there is some $x \in V$ such that both $v^{\max}$ and u have edges going to x. Then it is clear that x will cause u to become a non-candidate at some point of time or another.

For the sake of completeness, we do a more rigorous case-by-case analysis below. Recall that we have already established that $v^{\max}$ will always send a probe-message to x at some point of time or another.

- **Case 2(a) (u does not send a probe-message to x):** This implies that u became inactive before it could send a probe-message to x. But then u could have become inactive only if the if-clause of Line 15 of Algorithm 2 got satisfied at some point. Then u became a non-candidate too at the same time and therefore would not become a leader.
- **Case 2(b) (u sends a probe-message to x before $v^{\max}$ does):** Suppose u sends a probe-message to x at round i and $v^{\max}$ sends a probe-message to x at round i', where $1 \leq i < i' \leq \log n$. If x had seen an ID higher than ID_u up until round i, then x immediately informs u and u becomes a non-candidate. So suppose not. Then, after round i, x sets its local variables L_x and N_x to ID_u and u respectively. Let $j > i$ be the smallest integer such that x receives a probe-message from a neighbor u' at round j, where $ID_{u'} > ID_u$. Note that $v^{\max}$ will always send a probe-message to x, therefore such a u' exists. Then, after round j, x sets its local variables L_x and N_x to $ID_{u'}$ and u' respectively, and informs u of this change (please see Line 17 of Algorithm 2). u becomes a non-candidate at that point of time.
- **Case 2(c) (u and $v^{\max}$ each sends a probe-message to x at the same time):** Since $ID_{v^{\max}}$ is the highest ID in the network, L_x is assigned the value $ID_{v^{\max}}$ at this point, and x tells u about $L_x = ID_{v^{\max}} > ID_u$, causing u to become a non-candidate.

– **Case 2(d) (u sends a probe-message to x after $v^{\max}$ does):** Suppose $v^{\max}$ sends a probe-message to x at round i and u sends a probe-message to x at round i', where $1 \leq i < i' \leq \log n$. Then x sets its local variables L_x and N_x to $ID_{v^{\max}}$ and $v^{\max}$, respectively, after round i. So when u comes probing at round $i' > i$, x tells u about $L_x = ID_{v^{\max}} > ID_u$, causing u to become a non-candidate.

□

Lemmas 9 and 10 together imply that Algorithm 2 elects a unique leader (the node with the highest ID) and is therefore *correct*.

2.4.2 Message Complexity

Lemma 11. *At the end of round i, there are at most $\frac{n}{2^i}$ "active" nodes.*

Proof. Consider a node v that is active at the end of round i. This implies that the if-clause of Line 15 of Algorithm 2 has not so far been satisfied for v, which in turn implies that $ID_v > ID_{w_{v,j}}$ for $1 \leq j \leq 2^i - 1$, therefore none of

$$w_{v,1}, w_{v,2}, \ldots, w_{v,2^i-1}$$

is active after round i. Thus for every active node at the end of round i, there are at least $2^i - 1$ inactive nodes. We call this set of inactive nodes, together with v itself, the "kingdom" of v after round i, i.e.,

$$KINGDOM_i(v) \stackrel{\text{def}}{=} \{v\} \cup \left\{w_{v,1}, w_{v,2}, \ldots, w_{v,2^i-1}\right\}$$

$$\text{and } |KINGDOM_i(v)| = 2^i.$$

If we can show that these kingdoms are disjoint for two different active nodes, then we are done.

Proof by contradiction. Suppose not. Suppose there are two active nodes u and v such that

$$u \neq v \text{ and } KINGDOM_i(u) \cap KINGDOM_i(v) \neq \phi$$

(after some round i, $1 \leq i \leq \log n$). Let x be such that $x \in KINGDOM_i(u) \cap KINGDOM_i(v)$. Since an active node obviously cannot belong to the kingdom of another active node, this x equals neither u nor v, and therefore

$$x \in \left\{w_{v,1}, w_{v,2}, \ldots, w_{v,2^i-1}\right\} \cap \left\{w_{u,1}, w_{u,2}, \ldots, w_{u,2^i-1}\right\},$$

that is, both u and v have sent their respective probe-messages to x. Then it is straightforward to see that x would not allow u and v to be active at the same time.

For the sake of completeness, we do a more rigorous case-by-case analysis below. We assume that (without loss of generality) $ID_v > ID_u$.

- **Case** 1 **(u sends a probe-message to x before v does):** Suppose u sends a probe-message to x at round j and v sends a probe-message to x at round j', where $1 \leq j < j' \leq i$. If x had seen an ID higher than ID_u up until round j, then x immediately informs u and u becomes inactive. Contradiction.

  So suppose not. Then, after round j, x sets its local variables L_x and N_x to ID_u and u respectively. Let $k > j$ be the smallest integer such that x receives a probe-message from a neighbor u' at round k, where $ID_{u'} > ID_u$. Note that v sends a

probe-message to x at round j', where $j < j' \leq i$, and $ID_v > ID_u$. Therefore such a u' exists. Then, after round k, x sets its local variables L_x and N_x to $ID_{u'}$ and u' respectively, and informs u of this change (please see Line 17 of Algorithm 2). u becomes inactive at that point of time, i.e., after round k, where $k \leq j' \leq i$. Contradiction.

- **Case** 2 **(both** u **and** v **send a probe-message to** x **at the same time):** Suppose that u and v each sends a probe-message to x at the same round j, where $1 \leq j \leq i$. Since $ID_v > ID_u$, x has at least one neighbor u' such that $ID_{u'} > ID_u$. Therefore x would not set L_x to ID_u (or N_x to u), and x would inform u about that after round j, causing u to then become inactive. Contradiction.

- **Case** 3 **(**u **sends a probe-message to** x **after** v **does):** Suppose v sends a probe-message to x at round j and u sends a probe-message to x at round j', where $1 \leq j < j' \leq i$. Then x sets its local variables L_x and N_x to ID_v and v, respectively, after round j. So when u comes probing at round $j' > j$, x tells u about $L_x \geq ID_v > ID_u$, causing u to become inactive. Contradiction.

□

Lemma 12. *In round i, at most* 3*n messages are transmitted.*

Proof. In round i, each active node sends exactly 2^{i-1} probe messages, and each probe-message generates at most two responses (corresponding to Lines 17 and 21 of Algorithm 2). Thus, in round i, each active node contributes to, directly or indirectly, at most 3.2^{i-1} messages. The result immediately follows from Lemma 11. □

Since the algorithm runs for $\log n$ rounds, the total number of messages transmitted is at most $3n \log n$, and Theorem 4 immediately follows.

2.5 A Deterministic Lower Bound

In this section we show that $\Omega(n \log n)$ is a lower bound on the message complexity for solving leader election with any deterministic algorithm in diameter-two networks. Recall that [2] shows a lower bound of $\Omega(n \log n)$ for the special case of complete graphs (i.e., diameter-one networks), and we know from [80] that the message complexity becomes $\Omega(m)$ for (most) diameter-three networks. Thus, Theorem 5 completes the picture regarding the message complexity of leader election when considering networks according to their diameter.

More formally: For every deterministic algorithm that solves implicit leader election in all diameter-two networks, there exists some graph of diameter two where the said algorithm sends at least $\Omega(n \log n)$ messages. As usual, n is the number of nodes in the network.

In other words, we show that

Theorem 5. *There is no deterministic algorithm that solves leader election in every diameter-two network of n nodes with $o(n \log n)$ messages.*

Remark 4. In [2], the $\Omega(n \log n)$ message lower bound was showed for complete networks under the non-simultaneous wakeup model in synchronous networks. As we have verified in our private communications with Shay Kutten (Professor, Technion), the same message bound can be shown to hold in the simultaneous wake-up model as well under

the restriction that the number of rounds is bounded by a function of n. We will show a lower bound of $\Omega(n \log n)$ message complexity by reducing the problem of "leader election in complete graphs" to that of "leader election in graphs of diameter two". This reduction itself would take two rounds and $O(n)$ messages. Then, since the former is known to have $\Omega(n \log n)$ message complexity under the aforementioned constraints, the latter would have the same lower bound too (see Section 2.1.1).

In the remainder of this section, we prove Theorem 5.

Proof by reduction. Suppose $\mathscr{A}$ is a leader election algorithm that works for any graph of diameter two. Let $G = (V, E)$ be our input instance for the problem of "leader election in complete graphs", i.e., G is the complete graph on n nodes, say.

The Reduction. G sparsifies itself into a diameter-two graph (G', say, where $G' = (V, E')$, where $E' \subsetneq E$) on which $\mathscr{A}$ works thereafter. This sparsification takes $O(n)$ messages and a constant number of rounds (two, to be exact) and is done as follows.

- **Round 1:** Each node v chooses one of its neighbours w (any arbitrary one), say, and asks its ID. If $ID_w > ID_v$, then both v and w agree to "drop" that edge, i..e., it won't use that for communication in the subsequent simulation of $\mathscr{A}$. Otherwise v and w keep that edge.

  For $v \in V$, if v has $\lceil \frac{n}{2} \rceil$ or more edges removed, then v makes itself a "candidate".

- **Round 2:** The candidates from the previous round send their ID's to all the nodes in the network using edges of G. By Lemma 13, there can be at most two such nodes. Thus the total number of messages sent is still $O(n)$. Then each node

(including the candidates themselves) receives the ID's of up to two candidates and chooses the highest of them to be the ID of the leader.

If no such node exists which has had $\lceil \frac{n}{2} \rceil$ or more edges removed, then G' has diameter two (please refer to Lemma 14), and we run $\mathscr{A}$ on G'. $\mathscr{A}$ returns a leader on G' which makes itself the leader of G too, and informs all its neighbors. This takes $O(n)$ messages.

2.5.1 Proof of Correctness

Observation 4. E has at most $(n-1)$-edges more than E'.

Proof. Each node except the node with the highest ID drops at most one edge. The node with the highest ID drops no edge. □

Lemma 13. *For $n \geq 3$, there can be at most two nodes in G' that have had $\lceil \frac{n}{2} \rceil$ or more edges removed.*

Proof. A simple counting argument tells us that Lemma 13 follows from Observation 4.

We need to be careful with the ceiling operators and the floor operators here though. For the sake of completeness, we do a more rigorous case-by-case analysis (for the two cases when n is *even* and when n is *odd*, respectively) below.

- **Case 1: $n = 2k$ for some integer $k \geq 2$.**

  Proof by contradiction. Suppose that there are three or more nodes that have had $\lceil \frac{n}{2} \rceil = k$ or more edges removed each (either by themselves or by their neighbors). Let u, v, and w be three such nodes. Since an edge is removed only if

one of the incident nodes has a higher ID than the other, not all of (u,v), (v,w), and (w,u) can have been removed. Thus by a simple counting argument (by the "principle of inclusion-exclusion"), the total number of edges removed is at least $(3k-3)+1 = 3k-2 > 2k-1 = n-1$, which contradicts Observation 4.

- **Case 2: $n = 2k+1$ for some integer $k \geq 1$.**

  Proof by contradiction Suppose that there are three or more nodes that have had $\lceil \frac{n}{2} \rceil = k+1$ or more edges removed each (either by themselves or by their neighbors). Let u, v, and w be three such nodes. Since an edge is removed only if one of the incident nodes has a higher ID than the other, not all of (u,v), (v,w), and (w,u) can have been removed. Thus by a simple counting argument (by the "principle of inclusion-exclusion"), the total number of edges removed is at least $(3(k+1)-3)+1 = 3k+1 > 2k = n-1$, which contradicts Observation 4.

□

Lemma 14. *If no node exists in G' which has had $\lceil \frac{n}{2} \rceil$ or more edges removed, then G' has diameter two.*

Proof. Clearly G' is not of diameter one since the node with the smallest ID in V always drops at least one edge.

Now let us consider any pair of nodes u and v, say. We show that the distance between u and v cannot be greater than 2. This follows from this simple "pigeon hole principle" argument: If no node has had $\frac{n}{2}$ or more edges removed, then all nodes are of degree at least $\frac{n}{2}$; so for each pair of nodes there should be at least one common neighbour.

Of course, we need to be careful with the ceiling operators and the floor operators here. For the sake of completeness, we do a more rigorous case-by-case analysis (for the two cases when n is *even* and when n is *odd*, respectively) below.

- **Case** 1**:** $n = 2k$ **for some integer** $k \geq 2$**.**

  Since no node exists in G' which has had $\lceil \frac{n}{2} \rceil = k$ or more edges removed, every node in G' has degree at least $(n-1)-(k-1) = k$. Thus for any $u, v \in V$, if $(u,v) \notin E'$, then there are at least $k+k-(n-2) = 2$ nodes in $V \setminus \{u,v\}$ that are common neighbors to both u and v.

- **Case** 2**:** $n = 2k+1$ **for some integer** $k \geq 1$**.**

  Since no node exists in G' which has had $\lceil \frac{n}{2} \rceil = k+1$ or more edges removed, that implies that every node in G' has degree at least $(n-1)-k = k$. Thus for any $u, v \in V$, if $(u,v) \notin E'$, then there is at least $k+k-(n-2) = 1$ node in $V \setminus \{u,v\}$, which is a common neighbor to both u and v.

□

2.6 Conclusion

We settled the message complexity of leader election throughout the diameter spectrum, by presenting almost tight bounds (tight upto polylog (n) factors) for diameter-two graphs which were left open by previous results [81, 80]. Several open problems arise from our work.

1. Is it possible to show a high probability bound of $O(n)$ messages for randomized, implicit leader election that runs in $O(1)$ rounds? This will match the lower bounds, by closing the polylog(n) factor. It might be possible to improve the analysis of our randomized, implicit algorithm to show $O(n \log n)$ messages.

2. Coming to deterministic algorithms, another very interesting question is whether *explicit* leader election can be performed in $\tilde{O}(n)$ messages in diameter-two graphs *deterministically*.

3. The question of explicit leader election naturally begs the question whether *broadcast*, another fundamental problem in distributed computing, can be solved *deterministically* in *diameter-two* graphs with $\tilde{O}(n)$ messages and polylog (n) rounds if n is known.[10]

4. Removing the assumption of the knowledge of n (or showing that it is not possible) for deterministic, implicit leader election algorithms with $\tilde{O}(n)$ message complexity and running in $\tilde{O}(1)$ rounds is open as well.

[10] In contrast, we note that $\Omega(m)$ is a lower bound for deterministic leader election algorithms on graphs of *diameter at least three*, even if n is known [80].

Appendices

2.A Proof for Function Minima in Lemma 1

Proof. We define the Lagrangian as

$$\mathscr{L} \stackrel{\text{def}}{=} f(x_1, x_2, \ldots, x_n) - \lambda(\sum_{i=1}^{n} x_i - C) \tag{2.7}$$

where λ is the Lagrange multiplier. We can find the *critical points* of the Lagrangian by solving the set of equations

$$\frac{\partial f}{\partial x_i} = \lambda \frac{\partial \sum_{i=1}^{n} x_i}{\partial x_i} \text{ for } i = 1, 2, \ldots, n \tag{2.8}$$

and

$$\sum_{i=1}^{n} x_i = C \tag{2.9}$$

Simplifying Equation 2.8, we get

$$-\frac{\log x_i}{x_i^2} = \lambda \text{ for } i = 1, 2, \ldots, n \tag{2.10}$$

One possible (feasible) solution of Equations 2.10 and 2.9 is,

$$x_i = \frac{C}{n} \text{ for all } 1 \leq i \leq n \tag{2.11}$$

and

$$\lambda^* = -\frac{\log(\frac{C}{n})}{(\frac{C}{n})^2} \tag{2.12}$$

Let X^* be a vector of dimension n defined by $X^* \stackrel{\text{def}}{=} (\frac{C}{n}, \frac{C}{n}, \ldots, \frac{C}{n})$. Then we have already shown that X^* and λ^* together constitute *critical points* for the Lagrange function $\mathscr{L}$. We claim that X^* is also a local minimum for $f(x)$ under the constraint of Equation 2.9.

We show that by constructing the Bordered Hessian matrix H^B of the Lagrange function. Let $L^*_{ij} \stackrel{\text{def}}{=} \frac{\partial}{\partial x_j}(\frac{\partial \mathscr{L}}{\partial x_i})\Big|_{X^*}$, where $\mathscr{L}$ is the Lagrange function as defined in Equation 2.7. Then

$$H^B = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & L^*_{11} & L^*_{12} & \cdots & L^*_{1n} \\ 1 & L^*_{21} & L^*_{22} & \cdots & L^*_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & L^*_{n1} & L^*_{n2} & \cdots & L^*_{nn} \end{bmatrix}$$

We note that $L^*_{ii} = \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^*$ for all $1 \leq i \leq n$, and $L^*_{ij} = 0$ for all (i, j) such that $i \neq j$. Hence

$$H^B = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* & 0 & \cdots & 0 \\ 1 & 0 & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* \end{bmatrix}$$

We show that H^B is *positive definite* (which is a sufficient condition for X^* to be a local

minimum) by checking the signs of the leading principal minors. For any $1 \leq i \leq n$, $|H_i^B|$ is the determinant of a square matrix of dimension $i+1$, and is given by

$$|H_i^B| = \begin{vmatrix} 0 & 1 & 1 & \cdots & 1 \\ 1 & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* & 0 & \cdots & 0 \\ 1 & 0 & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* \end{vmatrix}$$

$$= -i(\frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^*)^{i-1}.$$

$$\text{But } \frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} > 0 \qquad \text{(since } C \geq n\sqrt{2},\ 2\log(\frac{C}{n})-1 > 0\text{)}$$

$$\text{and } \lambda^* = -\frac{\log(\frac{C}{n})}{(\frac{C}{n})^2} < 0.$$

Hence

$$\frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^* > 0$$

$$\implies -i(\frac{2\log(\frac{C}{n})-1}{(\frac{C}{n})^3} - \lambda^*)^{i-1} < 0$$

i.e., $|H_i^B| < 0$ for all $1 \leq i \leq n$

$\implies H^B$ is positive definite.

□

Chapter 3

Fast Byzantine Counting in Small-world Networks

We study the fundamental problem of counting the number of nodes in a sparse network (of unknown size) under the presence of a large number of Byzantine nodes[1]. We assume the full information model where the Byzantine nodes have complete knowledge about the entire state of the network at every round (including random choices made by all the nodes), have unbounded computational power, and can deviate arbitrarily from the protocol. Essentially all known algorithms for fundamental Byzantine problems (e.g., agreement, leader election, sampling) studied in the literature assume the knowledge (or at least an estimate) of the size of the network. It is non-trivial to design algorithms for Byzantine problems that work without knowledge of the network size, especially in bounded-degree (expander) networks where the local views of all nodes are (essentially)

[1]This chapter is based on joint work with Gopal Pandurangan and Peter Robinson; it contains material from [26].

the same and limited, and Byzantine nodes can quite easily fake the presence/absence of non-existing nodes. To design truly local algorithms that do not rely on any global knowledge (including network size), estimating the size of the network under Byzantine nodes is an important first step.

Our main contribution is a randomized distributed algorithm that estimates the size of a network under the presence of a large number of Byzantine nodes. In particular, our algorithm estimates the size of a sparse, "small-world", expander network with up to $O(n^{1-\delta})$ Byzantine nodes, where n is the (unknown) network size and δ can be any arbitrarily small (but fixed) positive constant. Our algorithm outputs a (fixed) constant factor estimate of $\log(n)$ with high probability; the correct estimate of the network size will be known to a large fraction ($(1-\varepsilon)$-fraction, for any fixed positive constant ε) of the honest nodes. Our algorithm is fully distributed, lightweight, and simple to implement. It runs in $O(\log^3 n)$ rounds and requires nodes to send and receive only small-sized messages[2] (per round). Any node's local computation cost per round is also small.

3.1 Introduction

Motivated by the need for robust and secure distributed computation in large-scale (sparse) networks such as peer-to-peer (P2P) and overlay networks, we study the fundamental *Byzantine counting* problem in networks, where the goal is to count (or estimate) the number of nodes in a network that can contain a large number of Byzantine nodes that can exhibit malicious behaviour. The Byzantine nodes can very well *collude* among themselves; all Byzantine nodes may even be under the control of an adversary who

[2] A "small-sized message" is one that contains a constant number of IDs and $O(\log n)$ additional bits.

strategically guides their behavior (possibly in order to foil the algorithm).

The Byzantine counting problem is challenging because the goal is to guarantee that most of the honest (i.e., non-Byzantine) nodes obtain a good estimate of the network size[3] despite the presence of a large number of Byzantine nodes (which have full information about all the nodes and can behave arbitrarily or maliciously) in the network. Byzantine counting is related to, yet different, compared with other fundamental problems in distributed computing, namely, *Byzantine agreement* and *Byzantine leader election*. Similar to the latter two problems, it involves solving a global problem under the presence of Byzantine nodes. However, it is a different problem, since protocols for Byzantine agreement or leader election do not necessarily yield a protocol for Byzantine counting. In a sense, the Byzantine counting problem can be considered to be more fundamental than Byzantine agreement and leader election, since many existing algorithms for these two problems (discussed below and in Section 3.1.3) assume knowledge of the number of nodes in the network n; some algorithms require at least a reasonably good estimate of n, typically a constant factor estimate of $\log n$. Indeed, one of the main motivations for this chapter is to design distributed protocols that can work with little or no global knowledge, including the network size. In this sense, an efficient protocol for the Byzantine counting problem can serve as a pre-processing step for protocols for Byzantine agreement, leader election and other problems that either require or assume knowledge of an estimate of n [6].

Byzantine agreement and leader election have been at the forefront of distributed computing research for several decades. The work of Dwork et al. [39] and Upfal [122]

[3] In sparse, bounded-degree networks, an adversary can always isolate some number of honest nodes; hence “almost-everywhere" knowledge is the best one can hope for in such networks (see [39]).

studied the Byzantine agreement problem in bounded-degree expander networks under the condition of *almost-everywhere* agreement, where *almost* all (honest) processors need to reach agreement as opposed to *all* nodes agreeing as required in the standard Byzantine agreement problem. Dwork et al. [39] showed how one can achieve almost-everywhere agreement with up to $\Theta(\frac{n}{\log n})$ of Byzantine nodes in a bounded-degree *expander* network (n is the network size). Subsequently, Upfal [122] gave an improved protocol that can tolerate up to a linear number of faults in a bounded degree *expander* of *sufficiently large spectral gap* (in fact, on *Ramanujan graphs*, which have the asymptotically largest spectral gap possible [57] — our protocol in this chapter also works on a similar type of expander). These algorithms require $O(\log n)$ rounds and polynomial (in n) number of messages; however, the local computation required by each processor is exponential. Both of the above algorithms *require knowledge of the global topology (including the knowledge of n)*, since at the start, nodes need to have this information hardcoded. The work of King et al. [71] was the first to study scalable (polylogarithmic communication and number of rounds, and polylogarithmic computation per processor) algorithms for Byzantine leader election and agreement. Similar to Dwork et al.'s and Upfal's algorithm, the nodes require hardcoded information on the network topology — which is also an *expander* graph — to begin with, including the network size. We note that the expansion property is crucially exploited in all the above works to achieve Byzantine agreement and leader election. Furthermore, the expander networks assumed in the works of Dwork et al. and Upfal are bounded-degree (essentially, regular) graphs, where without prior knowledge it is difficult for nodes to have a knowledge of the network size.

The works of [10], [7], and [8] studied stable agreement, Byzantine agreement, and

Byzantine leader election (respectively) in dynamic networks (see also [6]), where in addition to Byzantine nodes there is also adversarial churn. All these works assume that there is an underlying bounded-degree regular expander graph and *all nodes are assumed to have knowledge of n*. It was not clear how to estimate n without additional information under presence of Byzantine nodes in such networks where the local views of all nodes are (essentially) the same and limited, and Byzantine nodes can quite easily fake the presence/absence of non-existing nodes. In fact, the works of [6, 8] raised the question of designing protocols in expander networks that work when the network size is not known and may even change over time, with the goal of obtaining a protocol that works when nodes have strictly local knowledge. This requires devising a distributed protocol that can measure global network parameters such as size, diameter, average degree, etc. with Byzantine nodes in sparse networks.

3.1.1 Our Contributions

We introduce and study the problem of Byzantine Counting. Our goal is to design a distributed algorithm that guarantees, despite a large number of Byzantine nodes, that almost all honest nodes know a good estimate of the network size in a bounded degree, "small world" network. We are not aware of any prior work that studies *Byzantine* counting in the setting addressed here.

Before stating our result, we briefly describe the key ingredients of our network model (we refer to Section 3.2.1 for the full details). We assume a *sparse*[4] network

[4]We note that it is difficult to design a *dense* network in a way such that nodes cannot infer information about the network size from their node degree, which would make the counting problem significantly easier to solve.

that has constant bounded degree (essentially regular) and has *high expansion* and large *clustering coefficient*. In other words, it is a "small-world" network [40]. Expander graphs have been used extensively as candidates to solve the Byzantine agreement and related problems in bounded degree graphs (e.g., as discussed earlier, see [39, 64, 67, 69, 122]); the expander property proves crucial in tolerating a large number of Byzantine nodes. The high expansion of such graphs have been exploited in previous works as well, most notably by Upfal [122] to solve the Byzantine agreement (with knowledge of n). For the Byzantine counting problem, which seems harder, however, expansion by itself seems not to be sufficient; our protocol also exploits the high clustering coefficient of the network crucially (see Section 3.1.2).

We assume that up to $O(n^{1-\delta})$ nodes can be *Byzantine*, where $\delta > 0$ can be an arbitrarily small (but fixed) positive constant. We assume a strong adversarial *full information model* where the Byzantine nodes (who have unbounded computational power) are *adaptive*, in the sense that they know the entire states of all nodes at the beginning of every round (including the messages sent by them), including the random choices made by the nodes up to and including the current round as well as future rounds (in other words, they are *omniscient*).

We note that the Byzantine nodes can very well *collude* among themselves; all Byzantine nodes may even be under the control of an adversary who strategically guides their behavior (possibly in order to foil the algorithm). However, we note that the Byzantine nodes can communicate only using the edges of the network, i.e., they can send messages directly only to their neighbors.

In our network model, where nodes have constant bounded degree, most nodes, with

high probability, see (essentially) the same local topological structure even for a reasonably large neighborhood radius — see Lemma 16, and hence nodes do not have any a priori local information that can help them estimate the network size. In this setting, Byzantine nodes can easily fake the presence/absence of nodes — thus trying to foil the estimate of the honest nodes.

Our main contribution is a distributed algorithm (see Section 3.3) that estimates the size of the network, even in the presence of a large number of Byzantine nodes. In particular, our algorithm estimates the size of a sparse (constant degree) "small-world" network with up to $O(n^{1-\delta})$ (for any small positive constant δ) Byzantine nodes, where n is the (unknown) network size. Our algorithm outputs a (fixed) constant factor estimate of $\log n$ with high probability; the correct estimate of the network size will be known to $(1-\varepsilon)$-fraction (where $\varepsilon > 0$ is a small constant, say 0.1) of the honest nodes.[5]

Our algorithm is the first known, decentralized Byzantine counting algorithm that can tolerate a large amount of Byzantine nodes. It is fully-distributed, localized (does not require any global topological knowledge), lightweight, runs in $O(\log^3 n)$ rounds, and requires nodes to send and receive "small-sized messages" only.[6] Any node's computation cost per round is also logarithmic.

The given algorithm is a basic ingredient that can be used for the design of efficient distributed algorithms resilient against Byzantine failures, where the knowledge of the

[5]We call ε *the error parameter* — by changing its value (please refer to Line 5 in the pseudocode in Algorithm 3), we (the algorithm designer) can control exactly how large a fraction of the honest nodes would estimate $\log n$ correctly (i.e., get a constant-factor approximation of $\log n$). Theorem 6, which is the main result of this chapter, tells us that at most ε-fraction of the honest nodes would *fail* to get a constant factor approximation of $\log n$.

[6]A "small-sized message" is one that contains a constant number of IDs and $O(\log n)$ additional bits.

network size (a global parameter) may not be known a priori. It can serve as a building block for implementing other non-trivial distributed computation tasks in Byzantine networks such as agreement and leader election where the network size (or its estimate) is not known a priori.

3.1.2 Technical Challenges

The main technical challenge that we have to overcome is designing and analyzing distributed algorithms in the presence of Byzantine nodes in networks where (honest) nodes only have local knowledge, i.e., knowledge of their immediate neighborhood. It is possible to solve the counting problem exactly in networks without Byzantine nodes by simply building a spanning tree and converge-casting the nodes' counts to the root, which in turn can compute the total number of nodes in the network. A more robust and alternate way that works also in the case of *anonymous* networks is the technique of *support estimation* [10, 6] which uses *exponential* distribution (or alternately one can use a *geometric* distribution, see e.g., [80, 99]) to accurately estimate the network size as described below.

Consider the following simple protocol for estimating the network size that uses the geometric distribution. Each node u flips an unbiased coin until the outcome is `heads`; let X_u denote the random variable that denotes the number of times that u needs to flip its coin. Then, nodes exchange their respective values of X_u whereas each node only forwards the highest value of X_u (once) that it has seen so far. We observe that X_u is geometrically distributed and denote its global maximum by $\bar{X}$. For any u, $\Pr(X_u \geq 2\log n) = (\frac{1}{2})^{2\log n}$, and by taking a union bound, $\Pr(\bar{X} \geq 2\log n) \leq \frac{1}{n}$. Furthermore,

$$\Pr(\bar{X} < \tfrac{\log n}{2}) = (1 - (\tfrac{1}{2})^{\frac{\log n}{2}})^n \leq \exp(-\sqrt{n}).$$

It follows that each node forwards at most $O(\log n)$ distinct values (whp). After $O(D)$ rounds (where D is the network diameter), each node knows the value of $\bar{X}$, and sets that as its estimate of $\log n$. Due to the above bounds on $\bar{X}$ it follows that (whp) it is a constant factor estimate of $\log n$. The support estimation algorithm [10, 6] which uses the exponential distribution works in a similar manner.

The geometric distribution protocol fails when even just one Byzantine node is present. Byzantine nodes can fake the maximum value or can stop the correct maximum value from spreading and hence can violate any desired approximation guarantee. Hence a new protocol is needed when dealing with Byzantine nodes.

Prior localized techniques that have been used successfully for solving other problems such as Byzantine agreement and leader election such as random walks and majority agreement (e.g., [7, 8]) do not imply efficient (i.e., fast algorithms that uses small message sizes) algorithms for Byzantine counting. For instance, random walk-based techniques crucially exploit a uniform sampling of tokens (generated by nodes) after $\Theta(\text{mixing time})$ number of steps. However, the main difficulty in this approach is that the mixing time is unknown (since the network size is unknown) — and hence it is unclear a priori how many random walk steps the tokens should take. Similar approaches based on the return time of random walks fail due to long random walks having a high chance of encountering a Byzantine node. One can also use Birthday paradox ideas to try to estimate n (e.g., these have been tried in a non-Byzantine setting [48]); these also fail in the Byzantine case.

We note that one can possibly solve Byzantine counting if one can solve Byzan-

tine leader election;[7] however, all known algorithms for Byzantine leader election (or agreement) *assume a priori knowledge (or at least a good estimate) of the network size*. Hence we require a new protocol that solves Byzantine counting from "scratch". In our random network model, where most nodes, with high probability, see (essentially) the same local topological structure (and constant degree) even for a reasonably large neighborhood radius (see Lemma 16), it is difficult for nodes to break symmetry or gain a priori knowledge of n.[8]

Another approach is to try to estimate the diameter of the network, which, being $\Theta(\log n)$ for sparse expanders, can be used to deduce an approximation of the network size. Assuming that there exists a leader in the network, one way to do this is for the leader to initiate the flooding of a message and it can be shown that a large fraction of nodes (say a $(1-\varepsilon)$-fraction, for some small $\varepsilon > 0$) can estimate the diameter by recording the time when they see the first token, since we assume a synchronous network. However, this method fails since it is not clear, how to break symmetry initially by choosing a leader — this by itself appears to be a hard problem in the Byzantine setting without knowledge of n.

We now give a high-level intuition behind our protocol. The main idea is based on

[7]Informally, the idea is as follows. If one can elect a honest leader, then it can initiate flooding by sending a message to the entire network; any other node can set an estimate of $\log n$ as the round number when it sees the message for the first time. It can be shown that in a sparse expander, $n - o(n)$ nodes will have a constant factor estimate of $\log n$.

[8]We point out that with constant probability, in our network model, due to the property of the d-regular random graph, an expected constant number of nodes might have multi-edges — this can potentially be used to break ties; however, this fails to work with constant probability. In any case, such symmetry breaking will fail in symmetric regular graphs.

using the geometric distribution, but there are several technical obstacles that we need to tackle (see Section 3.3).

The algorithm operates in *phases*. In phase i, each honest node estimates the number of nodes at distance i (in particular, whether there are any nodes at all) by observing the maximum (or near-maximum) value, generated according to the geometric distribution, at distance i; this value can be propagated by flooding for exactly i steps (i.e., rounds). We only allow certain values to propagate in phase i; this *avoids congestion* and hence our algorithm works using only small message sizes. As i increases, i.e., when it becomes $a \log n$, for some small constant $0 < a < 1$, this provides a constant factor estimate of $\log n$. Up to a distance of $i = a \log n$, most nodes (i.e., $n - o(n)$ nodes) do not see any values from Byzantine nodes, since most nodes are a distance at least $a \log n$ from any Byzantine node — this is due to the property of the *expander* graph. However, as i increases, the Byzantine nodes can introduce fake values and hence can fool most of the nodes into believing that the network is much larger than it actually is. To overcome this, the protocol exploits the fact that nodes have high clustering coefficients — which implies that a node's neighbors are well-connected among themselves. Each (honest) node checks with its neighbors to see if the value sent by the Byzantine node is consistent among the neighbor set; if not, this (high) value is discarded.

There are significant complications in implementing this idea, since Byzantine nodes can lie about the identity of neighboring nodes. We refer to Section 3.3.3 for more details.

3.1.3 Other Related Works

There have been several works on estimating the size of the network, see e.g., the works of [58, 48, 114, 121, 86, 29], but all these works do not work under the presence of Byzantine adversaries. There has been some work on using network coding for designing byzantine protocols (see e.g., [62]); but these protocols have polynomial message sizes and are highly inefficient for problems such as counting, where the output size is small. There are also some works on topology discovery problems under Byzantine setting (e.g., [98]), but these do not solve the counting problem.

Several recent works deal with Byzantine agreement, Byzantine leader election, and fault-tolerant protocols in dynamic networks. We refer to [52, 10, 7, 5, 8] and the references therein for details on these works. These works crucially assume the knowledge of the network size (or at least an estimate of it) and don't work if the network size is not known.

There has been significant work in designing peer-to-peer networks that are provably robust to a large number of Byzantine faults [42, 56, 96, 111]. These focus only on (robustly) enabling storing and retrieving data items. The problem of achieving almost-everywhere agreement among nodes in P2P networks (modeled as expander graphs) is considered by King et al. in [71] in the context of the leader election problem; essentially, [71] is a sparse (expander) network implementation of the full information protocol of [69]. In another recent work [68], the authors use a spectral technique to "blacklist" malicious nodes leading to faster and more efficient Byzantine agreement. The work of [52] presents a solution for maintaining a clustering of the network, where each cluster contains more than two-thirds honest nodes with high probability in a setting where the

size of the network can vary polynomially over time. All these works assume an exact knowledge of or some good estimate of the network size and do not solve the Byzantine counting problem.

The work of [20] shows how to implement uniform sampling in a peer-to-peer system in the presence of Byzantine nodes where each node maintains a local "view" of the active nodes. We point out that the choice of the view size and the sample list size of $\Theta(n^{\frac{1}{3}})$ necessary for withstanding adversarial attacks requires the nodes to have a priori knowledge of a polynomial estimate of the network size. [58] considers a dynamically changing network *without* Byzantine nodes where nodes can join and leave over time and provides a local distributed protocol that achieves a polynomial estimate of the network size. In [21], the authors present a gossip-based algorithm for computing aggregate values in large dynamic networks (but without the presence of Byzantine failures), which can be used to obtain an estimate of the network size. The work of [30] focuses on the consensus problem under crash failures and assumes knowledge of $\log n$, where n is the network size.

Byzantine fault detection in the context of asynchronous distributed systems. There have also been several works on Byzantine fault detection — see, e.g., [4], [66], [54], and [51]. Alvisi et al. [4] consider the problem of fault detection in Byzantine *quorum systems* and design statistical methods to compute the current number of failures at any point of time. Their model assumes the knowledge of n, the total number of servers, whereas their goal is also clearly different.

Kihlstrom et al. [66] propose and analyze new classes of Byzantine fault detectors to solve the consensus problem in an asynchronous distributed system of n processes, in

which the number of (Byzantine-) faulty processors is strictly less than $\frac{n}{3}$. Haeberlen et al. [54] propose a new idea for Byzantine fault detection by achieving *eventual strong completeness* where every faulty node is eventually blacklisted by every correct node. The underlying communication graph is a *complete graph* in both the network models of [66] and [54], thus the knowledge of n becomes immediate and trivial.

Greve et al. [51] design and analyze a powerful Byzantine failure detector that works in dynamic distributed systems, where both the number of processors and the topology of the communication graph can change from round to round. Their work does not assume any knowledge of n; however, their work does *not* solve the Byzantine counting problem either — *no* estimate about the *global* network size can be made during the execution of their algorithm.

3.2 Preliminaries

3.2.1 Computing Model and Problem Definition

The distributed computing model. We consider a synchronous network represented by a graph G whose nodes execute a distributed algorithm and whose edges represent connectivity in the network. The computation proceeds in synchronous rounds and any message that is sent by some node u to its neighbors in some round $r \geq 1$ will be received by the end of round r.

Byzantine nodes. Among the n nodes (n or its estimate is not known to the nodes initially), up to $B(n)$ can be *Byzantine* and deviate arbitrarily from the given protocol.

Throughout this chapter, we assume that $B(n) = O(n^{1-\delta})$ (where n is the unknown network size), for any $\frac{3}{d} < \delta \leq 1$, where d is a degree parameter that is an even constant ≥ 8. We note that d is *not* the actual degree of a node — d is closely related to, but *different* from, the actual degree of a node in G. For more details, please see Section 3.2.1 below.

We say that a node u is *honest* if u is not a Byzantine node and use `Honest` to denote the set of honest nodes in the network.

Byzantine nodes are *adaptive*, in the sense that they have complete knowledge of the entire states of all nodes at the beginning of every round (including random choices made by all the nodes), and thus can take the current state of the computation into account when determining their next action (they also can know the future random choices of honest nodes). As noted before, the Byzantine nodes can very well *collude* among themselves; all Byzantine nodes may even be under the control of an adversary who strategically guides their behavior (possibly in order to foil the algorithm). The Byzantine nodes have unbounded computational power, and can deviate arbitrarily from the protocol. This setting is commonly referred to as the *full information model*.

We assume that the Byzantine nodes are *randomly distributed* in the network.

Distinct IDs. We assume that all nodes (including Byzantine ones) have *distinct IDs*, which they cannot fake while communicating with a neighbor. We assume that the n IDs are chosen from a large set that is unknown a priori to the nodes. In particular, this precludes most nodes from estimating $\log n$ by potentially looking at the length of their IDs.

Network Topology. Let $G = (V, E)$ be the graph representing the network. We take G to be the union of two other graphs H and L (both defined below). That is, $V(G) = V(H) = V(L) = V$, say, and $E(G) = E(H) \cup E(L)$. We take H to be a sparse, random d-regular graph that is constructed by the union of $\frac{d}{2}$ (assume $d \geq 8$ is an even constant) random Hamiltonian cycles of n nodes. We call this random graph model the $H(n,d)$ *random graph model*. It is known that such a random graph is an expander with high probability (see Lemma 27). The $H(n,d)$ random, regular graph model is a well-studied and popular random graph model (see e.g., [127]), and has been used as a model for peer-to-peer networks and self-healing networks [82, 103].

$E(L)$ is defined as follows. For $u, v \in V$, $(u,v) \in E(L)$ if and only if $dist(u,v) \leq k$ in H, where $k = \lceil \frac{d}{3} \rceil$ is a positive integer. We call k the *lattice parameter*.

In other words, each node has direct connections (via edges of L) to nodes that are within distance k. Note that adding the edges of L makes H a *"small-world"* network, i.e., for each node v in G, the neighbors of v within distance $\frac{k}{2}$ in H are connected to each other (thus the clustering coefficient is increased in G compared to H). The small-world property complements the expander property of the d-regular random graph, since the clustering coefficient of the random regular graph is small. We exploit both properties crucially in the protocol. The larger the degree d, the larger will be k, and the larger will be the robustness to Byzantine nodes, i.e., up to $O(n^{1-\delta})$ Byzantine nodes can be tolerated where $\frac{3}{d} < \delta \leq 1$ (as defined earlier). Our small-world network is inspired by and related to the Watts-Strogatz model [124, 18]. However, it is important to note that the Watts-Strogatz model allows some nodes to have $\Theta(\log n)$ degree whereas in our model, nodes can have only constant degree, i.e., every node has degree $O(1)$.

It is important to note that nodes in G do not know a priori which edges are in H and which are in L. However, as shown in Lemma 15, most nodes can distinguish between the two types of edges using a simple protocol.

We point out that even though we assume the specific type of network model described above, intuitively, the essentially identical local topological structure captures the most difficult aspects of "nearly-symmetric" networks. Moreover, our results can be extended to apply to potentially any graph that has high expansion and high clustering coefficient (e.g., one can presumably take any bounded-degree expander rather than a d-regular graph as H).

Problem and Goal. Our goal is to design a distributed protocol to estimate the number of nodes in G, even under the presence of a large number of Byzantine nodes. The problem is non-trivial, since each node has a local view and knowledge that is independent of the network size. We would like our protocol to run fast, i.e., run in polylogarithmic (in the unknown network size n) rounds, and to use only "small-sized" messages. A "small-sized message" is one that contains a constant number of IDs (i.e., $O(1)$ IDs) and $O(\log n)$ additional bits.

We now present the formal definition of the Byzantine counting problem. Since we assume a *sparse* (constant bounded degree) network and a large number of Byzantine nodes, it is difficult to ensure that every honest node eventually knows an exact estimate of n. This motivates us to consider the following "approximate, almost everywhere" variant of counting:

Definition 2 (Byzantine Counting)**.** *Suppose that there are $B(n)$ Byzantine nodes in the network and let ε be an arbitrarily small (but fixed) positive constant. We say that an*

algorithm solves Byzantine Counting in T rounds *if the following properties hold in all runs:*

1. *Every honest node u decides on an estimate of* $\log n$*, denoted by $\mathscr{L}_u$, within T rounds.*

2. *There is a set S of at least $(1-\varepsilon)n - B(n)$ honest nodes such that each $u \in S$ has a constant factor estimate of* $\log n$*; i.e., there are fixed positive constants c_1 and c_2 such that $c_1 \log n \leq \mathscr{L}_u \leq c_2 \log n$.*

3.2.2 Notations and a Few Basic Concepts

3.2.2.1 Distance Metrics

For any two nodes $u, v \in V$, $dist_G(u,v)$ denotes the *distance* between them in graph G, i.e., the length of a shortest path between u and v in G. Similarly, $dist_H(u,v) \stackrel{\text{def}}{=}$ the length of a shortest path between u and v in H; we follow the convention $dist_G(v,v) = dist_H(v,v) = 0$. For any node u and any set $V' \subset V(G) = V$, the distance between u and V' (in G) is defined as

$$dist_G(u,V') \stackrel{\text{def}}{=} min\{dist_G(u,v) \mid v \in V'\}.$$

For any two subsets V' and V'' of $V(G) = V$, the distance between V' and V'' (in G) is defined as

$$dist_G(V',V'') \stackrel{\text{def}}{=} min\{dist_G(u,v) \mid u \in V', v \in V''\}.$$

In all our notations, the subscript G or H denotes the underlying graph. We will, however, for the most part, talk about H and thus we omit the subscript H for simplicity.

For any $v \in V(H)$ and any positive integer r, the *radius r ball* $B(v,r)$ *of* v, is defined as the set of nodes within distance r from v, i.e.,

$$B(v,r) \stackrel{\text{def}}{=} \{w \in V(H) \mid 0 \leq dist(v,w) \leq r\}.$$

Similarly, the *radius r boundary* $Bd(v,r)$ *of* v is defined as the set of nodes at distance r from v (i.e., at the boundary), i.e,

$$Bd(v,r) \stackrel{\text{def}}{=} \{w \in V(H) \mid dist(v,w) = r\}.$$

Observation 5. In a d-regular graph, for any vertex v, the number of vertices that are within a τ-distance of v is bounded by $(d-1)^{\tau+1}$, i.e., $|B(v,\tau)| < (d-1)^{\tau+1}$.

The next observation follows since any two vertices that are within τ-distance of each other in G, are within $k\tau$ distance of each other in the d-regular graph H:

Observation 6. In the graph G, for any vertex v, the number of vertices that are within a τ-distance of v is bounded by $(d-1)^{k\tau+1}$, i.e., $|B_G(v,\tau)| < (d-1)^{k\tau+1}$.

Definition 3. *We call u a child of w with respect to v (or w the parent of u, with respect to v) if u is a child of w in the BFS tree rooted at v. Similarly, we call u and w siblings with respect to v if they are siblings in the BFS tree rooted at v. We note that, as is our usual custom, this BFS tree is in the graph H and not in G.*

It is important to keep in mind that nodes in G do not know a priori which edges are in H and which are in L. However, the following lemma assures us that most honest nodes can distinguish between the two types of edges using a simple protocol.

Lemma 15. *For any honest node v, if v has no Byzantine neighbor in G (that is, no Byzantine neighbor in its k-distance neighborhood in H), then v can faithfully reconstruct*

the topology of its k-distance neighborhood in H from the information provided by its G-neighbors.

Proof. For any $x \in V(G)$, let $N_G(x)$ denote the set of G-neighbors of x. Let w and u be two G-neighbors of v. Then we observe that

- w is a child of u with respect to v if and only if $N_G(w) \cap N_G(v) \subset N_G(u) \cap N_G(v)$.
- u is a child of w with respect to v if and only if $N_G(u) \cap N_G(v) \subset N_G(w) \cap N_G(v)$.
- u and v are siblings if $u \in N_G(w)$ and $w \in N_G(u)$ but neither of them is a child of the other.

□

Remark 5. Since d and k are constants, the list of neighbors is still $O(1)$, and hence can be exchanged in a constant number of rounds (using small sized messages).

3.2.2.2 The "Locally Tree-like" Property of an $H(n,d)$ Random Graph

The "locally tree-like" property of an $H(n,d)$ random graph says that for most nodes w, the subgraph induced by $B(w,r)$ up to a certain radius r looks like a tree. Let G be an $H(n,d)$ random graph and w be any node in G. Consider the subgraph induced by $B(w,r)$ for $r = \frac{\log n}{10 \log d}$. Let u be any node in $Bd(w,j)$, $1 \leq j < r$. u is said to be *typical* if u has only one neighbor in $Bd(w,j-1)$ and $(d-1)$-neighbors in $Bd(w,j+1)$; otherwise it is called *atypical*.

Definition 4 (Locally Tree-Like Property)**.** *We call a node w* locally tree-like *if no node in $B(w,r)$ is atypical. In other words, w is locally tree-like if the subgraph induced by $B(w,r)$ is a $(d-1)$-ary tree.*

Using the properties of the $H(n,d)$ random graph model and standard concentration bounds, it can be shown that most nodes in G are locally tree-like (see Section 3.A for more details):

Lemma 16. *In an $H(n,d)$ random graph, with high probability, at least $n - O(n^{0.8})$ nodes are locally tree-like.*

3.2.2.3 Classifying Different Kinds of Nodes in the Network

Definition 5. *We categorize the nodes in V into the following distinct categories and state some immediate bounds. Unlike our usual convention, the distances referred to in this definition refer to the respective distances in G (not in H, as is usual).*

1. ***Byzantine nodes:*** *The set of Byzantine nodes is denoted by $\texttt{Byz}$. $|\texttt{Byz}| = n^{1-\delta}$, by definition.*

2. ***Honest nodes:*** *The set of honest nodes is defined to be $\texttt{Honest} \stackrel{\text{def}}{=} V \setminus \texttt{Byz}$. $|\texttt{Honest}| = n - n^{1-\delta}$, by definition.*

3. ***Locally tree-like nodes:*** *Please refer to Definition 4. That is, the set of the locally tree-like nodes is defined as $\texttt{LTL} \stackrel{\text{def}}{=} \{v \in V \mid v \text{ is locally tree-like}\}$. $|\texttt{LTL}| \geq n - O(n^{0.8})$, by Lemma 16.*

4. ***Non-locally-tree-like nodes:*** *The set of the non-locally-tree-like nodes is defined as $\texttt{NLT} \stackrel{\text{def}}{=} V \setminus \texttt{LTL}$. $|\texttt{NLT}| \leq O(n^{0.8})$, by Lemma 16.*

5. ***Unsafe nodes:*** *The set of nodes that have one or more $\texttt{NLT}$ nodes within a distance of $a \log n$, where $a \stackrel{\text{def}}{=} \frac{\delta}{10k\log(d-1)}$. If we denote the set of unsafe nodes by $\texttt{Unsafe}$, then*

$$\texttt{Unsafe} \stackrel{\text{def}}{=} \{v \in V \mid dist_G(v, \texttt{NLT}) \leq a \log n\}.$$

$|\texttt{Unsafe}| \leq O(n^{0.8+\frac{\delta}{10}}) = o(n)$, *by Clause 4 of Definition 5, Observation 6, and Lemma 16.*

6. ***Safe nodes:*** *Nodes that are not unsafe. In other words, the set of nodes that have no* `NLT` *nodes within a distance of* $a \log n$*. If we denote the set of safe nodes by* `Safe`*, then*

$$\texttt{Safe} \stackrel{\text{def}}{=} \{v \in V \mid dist_G(v, \texttt{NLT}) > a \log n\}.$$

$|\texttt{Safe}| \geq n - O(n^{0.8+\frac{\delta}{10}}) = n - o(n)$, *by Clause 5 of Definition 5, Observation 6, and Lemma 16.*

7. ***Bad nodes:*** *The set of bad nodes is defined to be* $\texttt{Bad} \stackrel{\text{def}}{=} \texttt{Byz} \cup \texttt{NLT}$*.* $|\texttt{Bad}| \leq n^{1-\delta} + n^{0.8} \leq 2n^{1-\delta}$ *(assuming* $\delta \leq 0.2$*). This follows from the definition.*

8. ***Byzantine-Unsafe nodes:*** *The set of nodes that have one or more bad nodes within a distance of* $a \log n$*, where* $a \stackrel{\text{def}}{=} \frac{\delta}{10k \log(d-1)}$*. If we denote the set of Byzantine-unsafe nodes by* `BUS`*, then*

$$\texttt{BUS} \stackrel{\text{def}}{=} \{v \in V \mid dist_G(v, \texttt{Bad}) \leq a \log n\}.$$

$|\texttt{BUS}| \leq 2(d-1)n^{1-\frac{9\delta}{10}} = o(n)$ *(follows from Observation 6 and the definition of* `BUS`*)*

9. ***Byzantine-Safe nodes:*** *Nodes that are not Byzantine-unsafe. In other words, the set of nodes that have no bad nodes within a distance of* $a \log n$*. If we denote the set of Byzantine-safe nodes by* `Byz-safe`*, then*

$$\texttt{Byz-safe} \stackrel{\text{def}}{=} \{v \in V \mid dist_G(v, \texttt{Bad}) > a \log n\}.$$

$|\texttt{Byz-safe}| \geq n - 2(d-1)n^{1-\frac{9\delta}{10}} = n - o(n)$, *by definition.*

3.3 The Algorithm and Its Analysis

For the sake of exposition, we first describe and analyze the algorithm's behavior *free from* the influence of any Byzantine nodes; in other words, we will assume that all nodes honestly execute the protocol. We will discuss the effects the Byzantine nodes may have in Section 3.3.3.

3.3.1 Description of the Algorithm

Phases and Subphases. The algorithm operates in *phases* where, in the i^{th} phase, the nodes consider their current estimate of $\log n$ to be i. We reserve the letter i exclusively to denote the phase that the algorithm is presently in. A single phase i consists of several *subphases* that correspond to repetitions of the same random experiment (described below; see also Lines 7-16 of Algorithm 3). We will index the subphases by j, i.e., we will very frequently use the phrase "in the j^{th} subphase of the i^{th} phase". We note that in a synchronized network the current values of i and j are known to all nodes in each round.

At the very beginning (that is, even before phase 1 starts), every honest node v asks its neighbors in G for their own IDs and the IDs of their respective neighbors. We observe that this takes a constant number of rounds. From that neighborhood information of its neighbors, v tries to reconstruct the topology of its k-distance neighborhood in H. Lemma 15 tells us that this is possible when there are no Byzantine nodes. We discuss how this

step works in the presence of Byzantine nodes in Section 3.3.3 below.

We now describe the phases of the algorithm in more detail. The i^{th} phase consists of exactly α_i subphases (repetitions), where $\alpha_i \stackrel{\text{def}}{=} \lceil \frac{\log(\frac{1}{\varepsilon})+i+1-\log d}{(i-2)\log(d-1)} \rceil$ and ε is the *error parameter*. That is, ε allows us to control how large a fraction of honest nodes can obtain a correct estimate of $\log(n)$; see Line 5 of Algorithm 3). This is also reflected in Theorem 6, which tells us that at most ε-fraction of the honest nodes *fail* to obtain a constant factor approximation of $\log(n)$.

Color of a Token. In each subphase j of some phase i, every node sends out some *tokens* (containing some information) that is propagated through the network for i rounds. Every token circulating in the network has a *color*, which is added by the node v that generated the token. To this end, node v tosses an unbiased coin until it gets its first head (see Line 7 of Algorithm 3). If node v obtains the first head in the r^{th} trial, we call r to be the *color* of v (see Line 8 of Algorithm 3). Note that different nodes might obtain different colors for the same subphase j.

To ensure that Byzantine nodes cannot inject corrupted information, each node invokes the procedure verifyColors at the end of each of the rounds during which tokens are forwarded, which discards potentially corrupted tokens. (For now, we ignore verifyColors and assume that all colors are verified; we discuss its inner workings in Section 3.3.3.) Each keeps track of the verified colors it has received over the intermediate rounds, which are used in the estimation procedure described next.

Estimating $\log(n)$. When i is much smaller than $\log n$, it is likely that most nodes will receive their respective highest colored tokens in the last round of the subphase. In

contrast, when the phase number i is of the same order as $\log n$, most nodes are bound to see their respective highest colored tokens long before the last round. This provides a node with a way to determine when its estimate i is close to the actual value of $\log n$.

3.3.2 Analysis of the Algorithm (Without Byzantine Nodes)

In this section we show that the algorithm gives a $(\frac{b}{a})$-factor approximation of $\log n$ with high probability, where

$$a \stackrel{\text{def}}{=} \frac{\delta}{10k\log(d-1)}$$

and

$$b \stackrel{\text{def}}{=} \frac{4}{\log(1+\frac{h}{d})}.$$

The symbol h denotes the edge-expansion of H. Note that $0 < a < b < 1$. We recall that $n^{1-\delta}$ is the number of Byzantine nodes in the network G, and d is the uniform degree of graph H (see Section 3.2.1).

Observation 7. $b\log n \geq 2D(H)$, where $D(H)$ is the diameter of H.

A High-level Overview of the Proof. We break our analysis up into two different stages of the algorithm. First, observe that as long as there are no Byzantine nodes, the condition in Line 2 does not become true and thus all honest nodes continue to the first phase. We show that the following properties hold with high probability for different ranges of the phase number i.

Algorithm 3 Byzantine Counting for a given error parameter $\varepsilon > 0$. Code for node v.

Node v exchanges its adjacency lists with its neighbors and then uses the received information to classify its incident edges as being part of either H or L.

If v gets conflicting or contradictory information from two or more of its neighbors in G, node v stops executing the algorithm and remains mute.

for $i \leftarrow 1, 2, \ldots$ **do** ▷ i denotes the phase node v is in

    $FlagTerminate \leftarrow 1$

    $\alpha_i \leftarrow \left\lceil \frac{\log(\frac{1}{\varepsilon})+i+1}{\log d+(i-2)\log(d-1)} \right\rceil$

    for $j \leftarrow 1, 2, \ldots, \alpha_i$ **do** ▷ Phase i consists of α_i subphases; the subphases are indexed by j

        v tosses a fair coin until the outcome is heads in the r-th trial, for some $r \geq 1$.

        $c_{v,i} \leftarrow r$

        Flood the color $c_{v,i}$, along the edges of H only, for exactly i rounds:

        for $t = 1, 2, \ldots, i$ **do**

            Invoke verifyColors for all newly received colors; returns set of *verified colors* C_t.

            $k_t \leftarrow \max(C_t)$

        end for

        if $\forall t \in [1, i-1] : k_i > k_t$ and $k_i > \log(d(d-1)^{i-1}) - \log\log(d(d-1)^{i-1})$ **then**

            $FlagTerminate \leftarrow 0$

        end if

    end for

    if $FlagTerminate = 1$ **then**

        Decide i and terminate all for-loops. ▷ v accepts i as the estimate of $\log n$

    end if

end for

1. If $i < a\log n$, then at least a $(1-\varepsilon)$-fraction of the honest nodes do not accept i as estimate of $\log n$, and thus continue executing the algorithm. Thus, at most εn honest nodes might decide wrongly; however, they still continue to generate tokens as well as forward tokens generated by other nodes. (Recall that $\varepsilon > 0$ is an arbitrarily small constant fixed a priori.)

2. If $i = b\log n$, all but $o(n)$ of the remaining active nodes accept i to be the estimate of $\log n$. As before, they still continue to generate tokens as well as forward tokens generated by other nodes.

We cannot say which way a node will decide when $a\log n \leq i < b\log n$. The above two properties are, however, sufficient to give us an approximation factor of $\frac{b}{a} = \frac{40k\log(d-1)}{\delta\log(1+\frac{h}{d})}$.

3.3.2.1 When i Is Small: In Particular, when $i < a\log n$

For the sake of the analysis in this subsection only, we will consider only *safe nodes*, i.e., only those nodes in the set `Safe`.

For any non-empty $V' \subset V(G)$, $c_{V'}^{\max}$ is defined as $c_{V'}^{\max} \stackrel{\text{def}}{=} \{c_v \mid v \in V'\}$. Suppose $|V'| = n'$. Note that from the geometric distribution, we get $\Pr[c_{V'}^{\max} > 2\log n'] \leq \frac{1}{n'}$, and $\Pr[c_{V'}^{\max} \leq \log n' - \log\log n'] < \frac{1}{n'}$.

We recall that phase i consists of α_i subphases, and the subphases are indexed by j. Let $Failure(i,j)$ be the event that in the j^{th} subphase of the i^{th} phase, $\exists t < i$ such that $k_t \geq k_i$. That is, $Failure(i,j)$ is the event that in the j^{th} subphase of the i^{th} phase, the node v receives the maximum color in some round $t < i$. Then

Lemma 17. $\Pr[Failure(i,j)] < \frac{1}{d(d-1)^{i-2}}$.

Proof. We give a sketch of the main proof ideas here.

We derive a lower bound for $\Pr[Success(i,j)]$, where $Success(i,j)$ is the complement of the event $Failure(i,j)$. That gives us an upper bound for $\Pr[Failure(i,j)]$.

We observe that one of the ways $Success(i,j)$ can happen is if $\forall\ t < i$, k_t is small enough (smaller than some number k_1, say), and if $k_i > k_1$, the said number. We calculate the probabilities for those two events separately (from the properties of the *geometric distribution* and the *locally-tree-like property* of H and then combine them to calculate $\Pr[Success(i,j)]$. □

Since there are α_i subphases in the i^{th} phase and all those subphases are independent of each other, we can combine the individual subphase failure probabilities to compute the failure probability for the whole phase.

Lemma 18. $\Pr[a$ safe *node v makes a wrong decision in the* i^{th} *phase*$] < \frac{\varepsilon}{2^{i+1}}$.

Lemma 18 promises us that any individual node has a small probability of error when $i < a \log n$. So the expected number of nodes to make an error is also small. We, however, want to show a high probability bound on the number of nodes that make a mistake.

In order to show that, we proceed along the usual way of formulating an indicator random variable and then computing the expectation of the sum of the individual indicator random variables by using the principle of linearity of expectation. We show the high probability bound by using the method of *bounded differences* (Azuma's Inequality, more specifically).

Now to the formal description.

Let Y_i^v be an indicator random variable which is 1 if and only if v decides i to be a correct estimate of $\log n$. Lemma 18 shows that $Pr[Y_i^v = 1] < \frac{\varepsilon}{2^{i+1}}$. Now let

$$Y_i = \sum_{v \in V} Y_i^v.$$

That is, Y_i denotes the number of nodes that decide *wrongly* in the i^{th} phase. We recall once again that here we are interested only in the case where $i < a \log n$. Then Y_i cannot be too large, i.e., not too many nodes can decide wrongly in one phase.

Lemma 19. *$Pr[Y_i > \frac{n\varepsilon}{2^i}] < \frac{1}{n^4}$ if $i < a \log n$.*

Proof.

$$\begin{aligned} E[Y_i] &= E[\sum_{v \in V} Y_i^v] = \sum_{v \in V} E[Y_i^v] && \text{(by linearity of expectation)} \\ &= \sum_{v \in V} Pr[Y_i^v = 1] && \text{(since } Y_i^v \text{ is an indicator random variable)} \\ &< \sum_{v \in V} \frac{\varepsilon}{2^{i+1}} = \frac{n\varepsilon}{2^{i+1}} \end{aligned}$$

Two vertices v and w are independent if their i-distance neighborhoods do not intersect, i.e., if the distance between them is greater than $2i$. In other words, v going defective can affect only those vertices that are within a distance of $2i$ to v. The number of vertices that are within a $2i$ distance of v is at most $(d-1)^{(2i+1)}$. By the Azuma-Hoeffding Inequality [38, Theorem 5.8],

$$\begin{aligned} &Pr[Y_i - E[Y_i] \geq \frac{n\varepsilon}{2^{i+1}}] \leq \exp(-\frac{(\frac{n\varepsilon}{2^{i+1}})^2}{2n \cdot (d-1)^{2(2i+1)}}) = \exp(-\frac{n\varepsilon^2}{2^{2i+3}.(d-1)^{4i+2}}) \\ &= \exp(-\frac{n\varepsilon^2}{2^k}), \text{ say, where } k = 2i + 3 + (4i+2)\log(d-1). \end{aligned}$$

Since, $i < a\log n = \frac{\delta \log n}{10\log(d-1)}$,

$$\begin{aligned} k &< \frac{\delta \log n}{5\log(d-1)} + 3 + 2\log(d-1) + \frac{2\delta \log n}{5} \\ &= \frac{\delta \log n}{5\log(d-1)}(2\log(d-1)+1) + 2\log(d-1) + 3 \\ &< \log n - \log\log n - 2 - 2\log(\frac{1}{\varepsilon}). \end{aligned}$$

Thus

$$\begin{aligned} &Pr[Y_i - E[Y_i] \geq \frac{n\varepsilon}{2^{i+1}}] \leq \exp(-\frac{n\varepsilon^2}{2^k}) \leq \exp(-\frac{n\varepsilon^2}{2^{\log n - \log\log n - 2 - 2\log(\frac{1}{\varepsilon})}}) \\ &= \exp(-\frac{n\varepsilon^2}{\frac{n}{4\log n \cdot \frac{1}{\varepsilon^2}}}) = \exp(-4\log n) < \frac{1}{n^4}. \end{aligned}$$

But again $E[Y_i] < \frac{n\varepsilon}{2^{i+1}}$. Hence $Pr[Y_i > \frac{n\varepsilon}{2^i}] \leq Pr[Y_i - E[Y_i] \geq \frac{n\varepsilon}{2^{i+1}}] < \frac{1}{n^4}$. □

Now this is true for one particular phase i. Summing over all the phases (recall that we are concerned here only with the case $i < a\log n$), we get that the fraction of nodes that make a wrong decision cannot be more than

$$\sum_{i<a\log n} \frac{\varepsilon}{2^i} < \varepsilon,$$

and this is true with probability

$$> (1 - \sum_{i<a\log n} \frac{1}{n^4}) > (1 - \frac{1}{n^3}).$$

Thus we have

Lemma 20. *For Algorithm 1, the following holds with probability* $> 1 - \frac{1}{n^3}$*: While* $1 \leq i < a\log n$*, at most* ε*-fraction of the nodes decide wrongly, i.e., decide* i *to be a correct estimate of* $\log n$ *(where* ε *is any arbitrarily small but fixed positive constant).*

3.3.2.2 When $i = \Theta(\log n)$: In Particular, When $i = b \log n$

For the case $i = \Theta(\log n)$, our goal will be to show the following result:

Lemma 21. *The following holds with probability at least* $1 - \frac{1}{n^2}$*: If a node v is still active at the beginning of phase* $i = b \log n$*, then, by the end of this phase, v decides on* $b \log n$ *as its estimate of* $\log n$ *and terminates.*

The above lemma can be shown by showing that the following statement holds with probability at least $1 - \frac{1}{n^2}$: In all the $i\alpha_i$ subphases of phase i (where $i = b \log n$), it is always the case that $c_V^{\max} \leq 4 \log n - 1$, where $c_V^{\max} \stackrel{\text{def}}{=} \max \{c_v \mid v \in V\}$, i.e., the highest color generated in the network. (Recall that here we assume all nodes to be honest.)

3.3.3 Robustness Against Byzantine Nodes

Algorithm 3 contains two basic safety mechanisms for mitigating the impact of Byzantine nodes, namely Lines 2 and 11 in the pseudo code. We first describe how these mechanisms work and formally analyze them in Sec. 3.3.4.

- **Mechanism 1 (Line 2):** At the very beginning (that is, even before phase 1 starts), every honest node v asks its neighbors in G for their own IDs and the IDs of their respective neighbors. We observe that this takes a constant number of rounds. From the received neighborhood information, v tries to reconstruct the topology of its k-distance neighborhood in H. Recall that Lemma 15 tells us that this is possible when there are no Byzantine nodes.

  When there are Byzantine nodes, however, some of the neighborhood data that reach v can be corrupted. Line 2 ensures that v shuts itself down (i.e., it behaves

like a *crash failure*) if v receives inconsistent or conflicting data from two or more of its neighbors.

- **Mechanism 2 (Procedure verifyColors; Line 11):** For every color that v receives from a neighbor w, say, v checks (via the *lattice edges*, i.e., the edges of L) with all the nodes in $B(w, k-1)$ (this ball B is defined with respect to H) to verify that w indeed received that color via a legitimate path (up to a distance of $k-1$) from its $(k-1)$-distance neighbors in H. Note that, for colors received within the first t time-steps (in any j^{th} subphase of any phase i), when $1 \leq t \leq k-1$, an honest node v checks with nodes in the smaller ball $B(w,t)$ (instead of $B(w,k-1)$).

  Lemma 23 below guarantees that for any honest node v, the Byzantine nodes cannot fool v into believing the existence of a k-length chain, composed purely of Byzantine nodes, in its k-distance neighborhood in H. Thus it ensures that a Byzantine node is *not* able to inject arbitrary colors into the network without raising an alarm.

3.3.4 Analysis of the Algorithm Assuming Byzantine Nodes

3.3.4.1 Preliminaries — Some Useful Observations

Let `Crashed` be the set of honest nodes that shut themselves down at the very beginning of the algorithm (see Line 2 of the pseudocode in Algorithm 3). Let `Core` be the largest connected component in H induced by `Good`, where $\texttt{Good} \stackrel{\text{def}}{=} \texttt{Honest} \setminus \texttt{Crashed}$.

The next lemma restates a structural property of expander graphs that we require, namely that even after removing a large set of nodes from the graph, we can still find a

large expander in the remaining subgraph:

Lemma 22 (Lemma 3 in [9])**.** `Core` *has size at least* $n - o(n)$*. Moreover,* `Core` *is an expander with edge-expansion at least* γ*, where* $\gamma > 0$ *is a constant.*

Observation 8. In the graph H, with high probability, there is no chain of length $\geq k$ composed of Byzantine nodes only.

Proof. We have that $k = \lceil \frac{d}{3} \rceil$ and $\delta > \frac{3}{d}$, implying $k\delta > 1$. We assume that $k\delta = 1 + \delta'$ for a fixed positive constant δ'.

The number of possible k-length chains is upper-bounded by $n \cdot d^{k-1}$. We recall that the Byzantine nodes are randomly distributed in the network. Therefore, for any one such chain, the probability that it is composed purely of Byzantine nodes is $(\frac{n^{1-\delta}}{n})^k = n^{-k\delta}$. By the union bound, the probability that there is at least one chain made only of Byantine nodes is upper-bounded by

$$n \cdot d^{k-1} \cdot n^{-k\delta} = n \cdot d^{k-1} \cdot n^{-(1+\delta')} \qquad \text{(since } k\delta = 1 + \delta')$$

$= \frac{d^{k-1}}{n^{\delta'}}$, which is low probability for a fixed positive constant k.

□

Lemma 23. *With high probability, it holds that, for any* $v \in$ `Honest`*, the Byzantine nodes cannot make* v *believe that there is a chain of length* $\geq k$ *composed entirely of Byzantine nodes, without causing* v *to shut down (i.e., execute Line 2).*

Proof. Consider an honest node v. If v has no Byzantine neighbors in G, then v gets true neighborhood information from all its neghbors in G, and is thus able to accurately

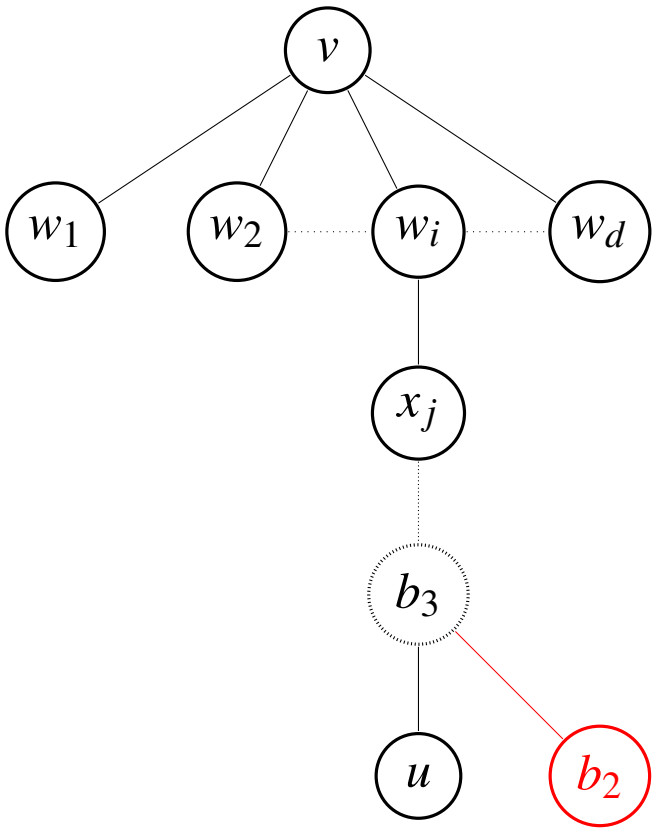

Figure 3.1: $\mathscr{C}_1 = (w_i, x_j, \ldots, b_3, b_2)$ is the k-length chain the Byzantine nodes are trying to tamper with. In reality, b_2 is *not* a child of b_3 (even though b_2 is directly connected to v in the graph G). So b_3 must hide the existence of a real child u, say, in order to concoct the existence of the fake, Byzantine child b_2.

reconstruct the exact topology of its k-distance neighborhood in H (please refer to Observation 15). Since H does not have any k-length chain of Byzantine nodes (please see Observation 8), v's reconstruction will have none either.

So suppose v has one or more Byzantine neighbors in G. Let $\mathscr{C}_1$ be the final k-length chain whose existence the Byzantine nodes are trying to "trick" v into believing. Since, in truth, $\mathscr{C}_1$ has at most $k-1$ Byzantine nodes (thanks to Observation 8), there must be a *dummy* node b_2, say, which the Byzantine nodes will try to insert into $\mathscr{C}_1$ (that is, to make it look as such in v's eyes).

Thus, b_3, the (fake) parent of b_2 in the chain $\mathscr{C}_1$, must report to v that it has b_2 as a child. While doing so, however, b_3 will need to suppress the existence of a (real) child u (which may or may not be Byzantine) because b_3 will need to maintain its degree d in H (in v's eyes).

But as u is directly connected to v in G, the Byzantine nodes cannot disrupt the communication between u and v. Since u knows b_3 to be its neighbor in H (we recall that b_3, even though Byzantine, cannot lie about its ID to u), the algorithm would dictate that u let it be known to v (regardless of whether or not u is Byzantine).

Therefore, v will have two conflicting pieces of information: it will hear from b_3 that b_3 and u are not neighbors in H, and v will hear the exact opposite from u. Thus, as per the algorithm, v will go into *crash failure*, i.e., will shut itself down (see Line 2 of the pseudocode in Algorithm 3. □

3.3.4.2 A High-level Overview of the Analysis

In the remainder of this section, we assume that there are $n^{1-\delta}$ Byzantine nodes in the network G. We will show that the algorithm gives a $(\frac{b}{a})$-factor approximation of $\log n$ with high probability, where $a \stackrel{\text{def}}{=} \frac{\delta}{10k\log(d-1)}$, $b \stackrel{\text{def}}{=} \frac{4}{\log(1+\frac{\gamma}{d})}$, and where γ is the edge-expansion of `Core`. Note that $0 < a < b < 1$, and d is the uniform degree of H. (H is a subset of G; for the exact definition of H see Section 3.2.1.)

Observation 9. $b\log n \geq 2D(\texttt{Core})$, where $D(\texttt{Core})$ is the diameter of `Core`.

Similarly to the case where we assumed all nodes to be honest (see Section 3.3.2), we break our analysis up according to the range of the phase counter i and show the following main result:

Theorem 6. *Let $\varepsilon > 0$ be an arbitrarily small (but fixed) positive constant. Algorithm 3 solves the Byzantine counting problem in $\Theta(\log^3 n)$ rounds with high probability, even in the presence of $O(n^{1-\delta})$ randomly distributed Byzantine nodes, where $\delta > 0$ is a small fixed constant that depends on the node degree d. Moreover, with high probability, all but an ε-fraction of the nodes in the network have a constant factor approximation of* $\log(n)$.

In the rest of this section we prove Theorem 6.

3.3.4.3 When i Is Small: In Particular, When $i < a\log n$

For the sake of the analysis in this subsection only, we will consider only *Byzantine-safe nodes*, i.e., only those nodes in the set `Byz-safe`.

We note that while $i < a\log n$, no token generated by a Byzantine node reaches a Byzantine-safe node (by the very definition of a Byzantine-safe node, as defined in

Definition 5). Thus for any $v \in \texttt{Byz-safe} \subset \texttt{Safe}$, the exact same analysis of Section 3.3.2.1 remains valid. That is, we have the same result, i.e., Lemma 20, as in the Byzantine-free setting.

3.3.4.4 When $i = \Theta(\log n)$: In Particular, When $i = b \log n$

We showed in Section 3.3.2.2 that the following statement holds with probability at least $1 - \frac{1}{n^2}$: If an honest node v is still active at the beginning of this phase, by the end of this phase, it accepts the current value of i, i.e., $b \log n$, to be a correct estimate of $\log n$ and terminates.

But the aforementioned analysis in Section 3.3.2.2 takes into account tokens generated by the honest nodes only. We still have to account for the Byzantine nodes, who can generate arbitrarily high colors.

Definition 6. *We say that a token is "high-colored" in phase i if it has a color larger than* $\log\left(d(d-1)^{i-1}\right) - \log\log\left(d(d-1)^{i-1}\right)$.

Finally, we recall that $k = \lceil \frac{d}{3} \rceil$ is a positive integer and is the *lattice parameter* (defined in Section 3.2.1).

Now we are ready to argue that even the Byzantine nodes are restricted by the structure of the network G. We show that a Byzantine node can inject a high-colored token into the network only at the beginning of a subphase. More specifically, we show the following lemma (by exploiting the structure of the network G):

Lemma 24. *The following statement holds with high probability: If a* core node *receives a high-colored token (generated by some Byzantine node) in round t in some subphase j, $1 \leq j \leq \alpha_i$, then $1 \leq t \leq k-1$.*

Proof. Suppose not. Suppose that there is at least one core node that receives a high-colored token in some subphase j, where $1 \leq j \leq \alpha_i$. Let $t \geq k$ be the earliest time-instant in that subphase when a core node receives a high-colored token. That is, there is some core node v that receives a high-colored token from a neighbor b, say, in the t^{th} round. Now b has to pretend that it received the high-colored token from somebody else (because b is not allowed to generate a token itself in the middle of a subphase). Since v has edges (the edges in L) to all the nodes in $B(b,k-1)$, v can contact all those nodes directly and check the veracity of b's claim. Since $t \geq k$, and since there are no Byzantine chains of length $\geq k$ (see Observation 8 and Lemma 23), there will be at least one honest node on any chain in $B(b,k-1)$ who would testify against b. □

This tells us that even if one or more Byzantine nodes introduce a high-colored token into the "good part" of the network (formally defined as the set `Core`), they are forced to do that early enough in a phase. Thus those high-colored tokens have ample time to propagate through `Core` (which is an expander) and reach all the *core nodes* "early enough". That violate the criterion for continuing on to the next phase (i.e., Line 15 in the pseudo code) and therefore the core nodes stop. That is, the core nodes accept the current value of i, which is $b\log n$, to be the correct estimate of $\log n$.

Lemma 25. *For $v \in$ `Core`, if v is still active at the beginning of phase i (when $i = b\log n$), then with high probability, by the end of this phase, it accepts the current value of i, i.e., $b\log n$, to be a correct estimate of $\log n$ and terminates.*

Proof. Lemma 24 says that the Byzantine nodes would not be able to push tokens into `Core` after the first $(k-1)$-rounds of a subphase without getting caught. So suppose that one or more Byzantine nodes introduce a sufficiently high color (so as to satisfy Line 15

of the pseudocode) into Core within the first $(k-1)$-rounds of some subphase j of the i^{th} phase, where $i = b\log n$.

But once even one core node receives a high color — Core being an expander (Please refer to Lemma 22) — that high color will start propagating through the network by means of flooding and will therefore reach every uncrashed node v within $D(\texttt{Core})$ rounds, where $D(\texttt{Core})$ is the diameter of Core. By Observation 9, this means that the highest color introduced by the Byzantine nodes will reach every core node v within $(\frac{b\log n}{2} + k - 1)$-rounds.

In other words, for any core node v, v will receive no higher color in round i than what it has already received before. This will violate the criterion for continuing, in particular, the variable $FlagTerminate$ will *not* be assigned the value 0 (see Line 15 of the pseudocode in Algorithm 3). Therefore v will accept the current value of i, which is $b\log n$, to be a correct estimate of $\log n$ and will terminate. □

Lemma 26. *If $i = b\log n$, then after the i^{th} phase, with high probability all but $o(n)$ of the nodes that were active at the beginning of this phase accept i as the estimate of $\log n$.*

Proof. Follows from Lemma 25 and Lemma 22. □

Lemma 26 together with Lemma 20 completes the proof of Theorem 6.

3.4 Conclusion and Open Problems

In this chapter, we took a step towards designing localized, secure, robust, and scalable algorithms for large-scale networks. We presented a fast (running in $O(\log^3 n)$ rounds) and lightweight (only simple local computations per node per round) distributed protocol

for the fundamental Byzantine counting problem tolerating $O(n^{1-\delta})$ (for any constant $\delta > 0$) Byzantine nodes while using only small-sized communication messages per round. Our work leaves many questions open.

A key open problem is to show a lower bound that is essentially tight with respect to the amount of Byzantine nodes that can be tolerated, or show an algorithm that can tolerate significantly more Byzantine nodes. Our protocol works only when the Byzantine nodes are randomly distributed; it would be good to remove this assumption and design a protocol that works under Byzantine nodes that are adversarially distributed. Another interesting question is whether one can improve the approximation factor of the estimate of $\log n$ to $1 \pm o(1)$.

Appendices

3.A The $H(n,d)$ Random Regular Graph Model: Definitions and Properties

In this section, we formally define the d-regular random graph model that we are assuming and also state and prove some crucial properties that we will use in the analysis.

3.A.1 Definitions

We assume a random regular graph that is constructed by the union of d random permutations as described below. Call such a random graph model the $H(n,d)$ model (or simply *H-graphs*). This model was also used by Law and Siu [82] to model Peer-to-Peer networks. A random graph in this model can be constructed by picking $\frac{d}{2}$ (assume d is even) Hamilton cycles independently and uniformly at random among all possible Hamilton cycles on the set of n vertices, and taking the union of these Hamilton cycles. This construction yields a random d-regular graph (henceforth called a $H(n,d)$ graph) that can be shown to be an expander with high probability (see Lemma 27). Note that an $H(n,d)$ graph is a d-regular multigraph whose set of edges is composed of the $\frac{d}{2}$ Hamil-

ton cycles. Friedman's [46] result below (rephrased here for our purposes) shows that a $H(n,d)$ graph is an expander (in fact, a *Ramanujan Expander*, i.e., the second smallest eigenvalue for these random graphs is close to the best possible) with high probability.

Lemma 27 ([46, 82])**.** *A random n-node, d-regular $H(n,d)$-graph (say, for $d \geq 6$) is an expander with high probability.*

3.A.2 Properties

We next show some basic properties of the $H(n,d)$ random graph which are needed in the analysis. We show some bounds on the sizes of $B(w,r)$ and $Bd(w,r)$.

Lemma 28. *1.* $|Bd(w,r)| \leq (d-1)|Bd(w,r-1)|$.

2. Whp $|Bd(w,r)| \geq (d-1-o(1))|Bd(w,r-1)|$, *for* $1 < r < \frac{\log n}{2\log d}$.

3. For some constant c and c′, $c'(d-1)^r \leq |B(w,r)| \leq c(d-1)^r$, *whp*

4. $|B(w,r)| = \Theta(|Bd(w,r)|)$.

Proof. Since the degree of each node is d, (1) follows. From (1) it is easy to show the upper bound on $|B(w,r)|$ in (3).

We next show (2).

We first bound the expected number of neighbours that a node $u \in Bd(w,r-1))$ has in $B(w,r-1)$. The expected number of neighbors of u in $B(w,r)$ is $\frac{(d-1)(n-|B(w,r-1)|)}{n} \leq (d-1)(1-\frac{\sqrt{n}}{n})$, since $|B(w,r-1)| < d^{\frac{\log n}{2\log d}} = \sqrt{n}$. Hence the expected number of nodes in $Bd(w,r)$ is $|Bd(w,r-1)|(d-1)(1-\frac{\sqrt{n}}{n})$. The high probability bound can be obtained via a Chernoff bound (one can consider the choices made by individual nodes as essentially

independent if one regards the "sampling without replacement" due to the permutations. This can be done if one pretends that the sample is from a set of size $n - \sqrt{n}$ (instead of n). This will not make a difference asymptotically.

The lower bound of (3) follows from (2) and (4) follows from (1), (2), and (3). □

Next we establish the "locally tree-like" property of an $H(n,d)$ random graph: For most nodes w, the subgraph induced by $B(w,r)$ up to a certain radius r looks "like a tree". This is stated more precisely as follows.

Definition 7. *Let G be an $H(n,d)$ random graph and w be any node in G. Consider the subgraph induced by $B(w,r)$ for $r = \frac{\log n}{10 \log d}$. Let u be any node in $Bd(w,j)$, $1 \leq j < r$. u is said to be "typical" if u has only one neighbor in $Bd(w,j-1)$ and $(d-1)$-neighbors in $Bd(w,j+1)$; otherwise it is called "atypical".*

Definition 8. *For $r = \frac{\log n}{10 \log d}$, we call a node w "locally tree-like" if no node in $B(w,r)$ is atypical. In other words, w is "locally tree-like" if the subgraph induced by $B(w,r)$ (where $r = \frac{\log n}{10 \log d}$) is a $(d-1)$-ary tree.*

The following lemma shows that most nodes in G are locally tree-like.

Lemma 29. *In an $H(n,d)$ random graph, with high probability, at least $n - O(n^{0.8})$ nodes are locally tree-like.*

Proof. Consider a node $w \in V$. We upper bound the probability that a node in $B(w,r)$, where $r = \frac{\log n}{10 \log d}$, is atypical. For any $1 \leq j < r$,

$$\Pr(u \in B(w,j) \text{ is atypical}) \leq (d-1) \cdot \frac{|B(w,j)|}{n} = O(\tfrac{1}{n^{0.9}}),$$

using the bound that $|B(w,j)| \leq d^r$ (the above gives an upper-bound on the probability that u has more than one neighbor in $B(w,j)$, in which case it is atypical). Hence the probability that there is some node u that is atypical in $B(w,r)$ is $O(\frac{n^{0.1}}{n^{0.9}}) = O(\frac{1}{n^{0.8}})$. Thus the probability that node w is not locally tree-like is at most $O(\frac{1}{n^{0.8}})$.

Let the indicator random variable X_w indicate the event that node w is locally tree-like. Let random variable $X = \sum_{w \in V} X_w$ denote the number of nodes that are locally tree-like. By linearity of expectation, using the above probability bound, it follows that the expected number of nodes in G that are not locally tree-like is at most $O(n^{0.2})$; in other words, $E[X] \geq n - O(n^{0.2})$.

To show concentration of X, we use Azuma's inequality [38, Theorem 5.3] as follows. Changing the value of X_w affects only the nodes within radius $r' = 2r = \frac{2\log n}{10\log d}$, i.e., at most $n^{0.2}$ nodes and hence affects $E[X]$ by at most $n^{0.2}$. Thus, we have

$$\Pr(|X - E[X]| > n^{0.8}) \leq 2\exp(-\frac{n^{\frac{8}{5}}}{n \times n^{\frac{2}{5}}}) = 2\exp(-n^{\frac{1}{5}}).$$

Hence, with high probability, at least $n - O(n^{0.8})$ nodes are locally tree-like. □

We now show a property that will be useful in our analysis; this follows immediately from the definition of locally-tree like and the regularity of the graph.

Corollary 1. *Let G be an $H(n,d)$ random graph and consider a node w in G. Assume that w is locally tree-like, i.e., the subgraph induced by $B(w,r)$ (where $r = \frac{\log n}{10\log d}$) is a tree. For every neighbor u of w, the respective subtrees rooted at u (in the subgraph induced by $B(w,r)$) are isomorphic; in particular each is a $(d-1)$-ary tree.*

Chapter 4

Fast Byzantine Routing in Dynamic Networks

Motivated by the need for designing secure, robust, and efficient fully-distributed computation in highly dynamic networks such as Peer-to-Peer (P2P) networks, we study distributed protocols for constructing and maintaining dynamic network topologies that guarantee efficient and reliable communication even in the presence of a large number of Byzantine (bad) nodes and a continual heavy adversarial churn (i.e., nodes joining and leaving the network continually over time). We assume that the Byzantine nodes have unbounded computational power, can behave arbitrarily and maliciously, and may collude among themselves. However, they are oblivious to the communication between good nodes (i.e., we assume private channels, but no other cryptographic assumptions). We also assume that an adversary controls the churn — it 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 topology changes from round to round).

Our main contribution is a randomized distributed protocol that guarantees with high probability the construction and maintenance of a sparse (logarithmic degree) distributed hash table (DHT) network that guarantees efficient, robust, and reliable communication between (almost all pairs of) good nodes even in the presence of $\frac{n}{\text{polylog}(n)}$ number of Byzantine nodes and *continual heavy* churn of up to $\frac{n}{\text{polylog}(n)}$ *adversarially* chosen nodes joining and leaving every round (where n is the stable network size). Our protocol is efficient, lightweight, and scalable, and it incurs only $\text{polylog}(n)$ overhead for topology construction and maintenance: only polylogarithmic (in n) bits need to be processed and sent by each node per round and any node's computation cost per round is also polylogarithmic in n. To our knowledge, this is the first protocol that can guarantee the above properties with such a large number of Byzantine nodes in a highly dynamic setting. In particular, our protocol improves over the protocol of Augustine et al. [9], that constructed and maintained a sparse expander network under similar high adversarial churn, but could not handle Byzantine nodes nor guaranteed efficient and robust communication.

4.1 Introduction

Peer-to-peer (P2P) computing has emerged as one of the key networking technologies with many application systems, e.g., Skype, BitTorrent, Symform, Bitcoin and other blockchain systems. For example, systems such as Symform [119] and BitTorrent Sync (Resilio) [19] are P2P-based storage services that allow data to be stored and retrieved among peers [104]. Recently, there has been a number of systems such as Storj [117], Filecoin [44], and Sia [115] that offer decentralized P2P cloud storage based on blockchain

technology.

Such peer-based data sharing avoids centralized storage and retrieval and is inherently scalable. However, some of these systems are not fully P2P; they also use dedicated centralized servers in order to guarantee high availability of data — this is necessary due to the highly dynamic and unpredictable nature of P2P. Furthermore, these systems can be susceptible to corruption due to presence of malicious peers. Consider the example of Bitcoin — a fully-decentralized P2P-based digital money with no central authority or middlemen [125]. A crucial aspect of Bitcoin and other blockchain systems is a computational mechanism that allows fault-tolerant (Byzantine) agreement among all honest nodes to agree on all the transactions despite the dynamism and presence of malicious nodes in the system. This is a costly operation (called mining) that only nodes with very high computational power can perform; furthermore there is a non-trivial probability of failure leading to corruption of the system. Hence, distributed computation in P2P systems must be robust to dynamism and failures, and also attack-resistant, besides being efficient and scalable.

P2P networks are highly dynamic networks characterized by a high degree of *churn*, in which nodes continually join and leave the network. Measurement studies of real-world P2P networks [41, 53, 113, 118] show that the churn rate is quite high: nearly 50% of peers in real-world networks can be replaced within an hour.[1] P2P algorithms have been proposed for a wide variety of tasks such as data storage and retrieval [104, 37, 36, 24, 55], collaborative filtering [22], spam detection [32], data mining [33], worm detection and suppression [89, 123], privacy protection of archived data [49], and recently,

[1]However, despite a large churn rate, these studies also show that the total number of peers in the network is relatively *stable*.

for cloud computing services as well [117, 44, 115, 16, 126]. However, all algorithms proposed for these problems have no theoretical guarantees of being able to work in a dynamically changing network with a very high churn rate and presence of Byzantine nodes. This is a major bottleneck in the wide-spread use of P2P systems.

There has been extensive research on P2P networks based on *Distributed Hash Table* or *DHT* schemes (e.g., [108, 116, 90]) which have also been implemented (to varying extents) in real-world P2P networks (e.g., [45, 126, 91]). A DHT creates a fully-decentralized addressable network that maps data items to peers and allows a peer to store and search a data item very efficiently (typically logarithmic in the size of the network) without flooding. Fully-decentralized DHT schemes are difficult to realize in real-world P2P networks, mainly due to the fact that it is not easy to maintain a DHT in a highly dynamic and Byzantine setting. In addition to this, since data is stored typically in some arbitrary node(s), fault-tolerance to node deletions is essential. Efficient search also requires that DHT routing operates correctly and efficiently despite the presence of Byzantine nodes. These are some of the reasons why DHT schemes, despite their efficient search and storage mechanism, have been somewhat less successful when it comes to practical deployment. Hence, it is of both theoretical and practical interest to develop simple and efficient DHT schemes that work provably well under high churn rates and presence of large number of Byzantine nodes.

Performing efficient distributed computation in highly dynamic networks where both nodes and edges can change continually by a large amount is a major challenge. Indeed, when the churn rate is linear (or close to linear) in the network size, in a *constant* number of rounds the entire network can be renewed! We would like distributed algorithms to

work correctly and efficiently in such networks that keep changing continually over time (not assuming any eventual stabilization or quiescence — unlike much of the earlier work on dynamic networks, e.g., [1, 35, 47, 3, 12, 15]).

Besides dynamism, another major challenge is dealing with malicious (also called *Byzantine*) nodes in the network which can try to foil the distributed protocol executed by honest (good) nodes. Byzantine protocols are at the heart of secure and robust protocols that can tolerate the presence of malicious nodes in a distributed system, such as a P2P network, which allows a large number of peers to enter the network with little or no admission control. Such malicious peers acting alone or in collaboration can cause disruption of service in P2P systems [93, 31, 84]. For a more recent and detailed account of distributed denial of service attacks (DDoS attacks), please see [23] and the references therein.

Designing scalable Byzantine protocols that work in highly dynamic networks is significantly more challenging compared to static networks. Indeed, until recently, almost all the work known in the literature (see e.g., [39, 64, 67, 70, 122, 72]) has addressed the fundamental Byzantine agreement and leader election problems only in static (bounded-degree) networks, and these approaches *do not work* for dynamic networks with changing topology. Such approaches fail in dynamic networks where both nodes *and* edges can change by a large amount in *every* round. For example, the work of Upfal [122] showed how one can achieve almost-everywhere Byzantine agreement[2] under up to a *linear* number — up to εn, for a sufficiently small $\varepsilon > 0$ — of Byzantine faults in a bounded-degree expander network (n is the network size). The algorithm requires $O(\log n)$ rounds

[2]In sparse, bounded-degree networks, an adversary can always isolate some number of non-faulty nodes, hence almost-everywhere is the best one can hope for in such network [39].

and polynomial (in n) number of messages; however, the local computation required by each processor is exponential. Furthermore, the algorithm requires knowledge of the global topology, since at the start, nodes need to have this information "hardcoded".

The work of King et al. [72] is important in the context of P2P networks, as it was the first to study scalable (polylogarithmic communication and number of rounds) algorithms for Byzantine agreement and leader election. However, as pointed out by the authors, their algorithm works only for static networks; here also the nodes require hardcoded information on the network topology to begin with, and thus the algorithm does not work when the topology changes. In fact, this work ([72]) raised the open question of whether one can design Byzantine agreement protocols that can work in highly dynamic networks with a large churn rate.

Motivated by the above considerations, several recent papers have taken steps towards designing provably efficient and robust algorithms for solving fundamental distributed computing problems in highly dynamic (with high churn) peer-to-peer networks [10, 7, 8, 5]. The work of [10] studies the fundamental distributed agreement problem in dynamic P2P networks with churn. Its main contribution is an efficient and scalable randomized distributed algorithm (i.e., each node processes and sends only polylogarithmic bits per round, and local computation per node is also lightweight) that guarantees stable, almost-everywhere agreement with high probability even under very high *adversarial* churn rate (up to linear in *n per round*, where n is the network size) in a *polylogarithmic* number of rounds. The work of [7] presented an efficient distributed algorithm for Byzantine agreement that works despite the presence of Byzantine nodes and high adversarial churn. This algorithm can tolerate up to $\frac{\sqrt{n}}{\text{polylog}\,(n)}$ Byzantine nodes and up to $\frac{\sqrt{n}}{\text{polylog}\,(n)}$ churn

per round, takes a polylogarithmic number of rounds, and is scalable. The work of [8] presented an efficient distributed algorithm for Byzantine leader election that can tolerate up to $O(n^{\frac{1}{2}-\varepsilon})$ Byzantine nodes (for any fixed positive constant ε) and up to $\frac{\sqrt{n}}{\text{polylog}(n)}$ churn per round, and that takes a polylogarithmic number of rounds. The work of [5] focused on the problem of storing, maintaining, and searching data in P2P networks. It presented a storage and maintenance algorithm that guarantees (with high probability) that data items can be efficiently stored (with only $O(1)$ copies of each data item) and maintained in a dynamic P2P network with churn rate up to $\frac{n}{\text{polylog}(n)}$ *per round*. However, this storage algorithm does *not* work in the presence of Byzantine nodes.

A crucial ingredient that underlies all the above results is the *assumption* that though the topology — both nodes and edges — can change arbitrarily from round to round and is controlled by an adversary, the topology in *every round* is an (bounded-degree) *expander* graph. In other words, the adversary is allowed to control the churn as well as change the topology with the *crucial restriction* that it always remains an expander graph in every round. The assumption of an ever-present underlying expander facilitates the design of efficient algorithms that are highly robust (tolerating a large amount of churn and Byzantine nodes per round).[3] However, this is a very strong assumption, *that itself needs to be satisfied if one desires truly distributed protocols that work under little or no assumption on the underlying topology*. This motivates designing a distributed protocol that actually *maintains* an expander topology under the presence of high adversarial continual churn and under presence of large number of Byzantine nodes. Expanders have

[3]The expander property ensures that a linear number of Byzantine nodes cannot partition the network into many small pieces; a large giant component still remains even under adversarial placement of Byzantine nodes — see [9, Lemma 3].

been used extensively to model dynamic P2P networks in which the expander property is preserved under insertions and deletions of nodes (e.g., see [102, 82, 101] and the references therein). However, none of these works guarantee the maintenance of an expander under the more challenging situation of high continual adversarial churn and presence of large number of Byzantine nodes. This is a fundamental ingredient that is needed to enable the applicability of previous results ([10, 7, 5, 8]) under a more realistic setting.

To address the above problem, the work of Augustine et al. [9] presented an efficient randomized distributed protocol that guarantees the maintenance of a *sparse* (bounded degree) topology with *high expansion* despite the presence of a large adversarial churn. This protocol can tolerate a churn rate of up to $\frac{n}{\text{polylog}(n)}$ per round (where n is the stable network size). The protocol is efficient, lightweight, and scalable, and it incurs only $\text{polylog}(n)$ overhead for topology maintenance: only polylogarithmic (in n) bits needs to be processed and sent by each node per round and any node's computation cost per round is also polylogarithmic. This was the first-known, fully-distributed protocol that guarantees expansion maintenance under highly dynamic and adversarial settings. However, a major drawback of this protocol is that it *does not work* in the presence of Byzantine nodes; even the presence of a small number (say $O(\log n)$) of Byzantine nodes can cause the protocol to fail. Furthermore, the expander network constructed by the above protocol (in the absence of Byzantine nodes) does not support efficient communication between nodes, i.e., low-cost routing in logarithmic number of steps. In particular, it does not construct and maintain a DHT. Similarly, the recent work of [11] gives a DHT scheme that is tolerant to large adversarial churn, but cannot handle

Byzantine nodes.

To summarize, the state of the art is that all known Byzantine protocols either assume only static settings or do not handle a large number of Byzantine nodes under highly dynamic settings. Furthermore, there is no known DHT scheme that can provably handle large adversarial churn and large number of Byzantine nodes.

4.1.1 Our Main Result

Our main result is an efficient randomized distributed protocol that guarantees the construction and maintenance of a *sparse*[4] addressable peer-to-peer network that guarantees secure, robust, and efficient communication between almost all pairs of good (honest) nodes despite the presence of a large adversarial churn and a large number of Byzantine nodes. The number of node changes *per round* is called the *churn rate* or *churn limit*. Though the churn rate can be large, we assume that the total number of nodes in the network is stable. We consider a churn rate of up to $\frac{n}{\text{polylog}(n)}$, where n is the stable network size. We assume that the churn is controlled by an adversary that has complete knowledge of what nodes join and leave and at what time, but is oblivious to the random choices made by the algorithm. The adversary can remove any set of nodes up to the churn limit in *every* round. At the same time, it should add an equal amount of[5] nodes to the network with the following restrictions:

[4] each node having only $\text{polylog}(n)$ degree

[5] Similar assumptions have been made in prior works that also assume a stable network size [10, 7, 5, 8, 9]. Our protocol can be adapted to work correctly as long as the number of nodes is reasonably stable over time, say, between $n - \xi n$ and $n + \xi n$ for some suitably small constant ξ (for the same amount of churn per round, i.e., $\frac{n}{\text{polylog}(n)}$ churn per round).

1. a new node should be connected to an existing node, and

2. the number of new nodes added to any one existing node should not exceed $O(1)$ per round.

The number of Byzantine nodes that can be present in the network at any round can be up to $\frac{n}{\text{polylog}(n)}$. We assume that the Byzantine nodes have unbounded computational power, can behave arbitrarily and maliciously, and may collude among themselves, but are oblivious to the communication between good nodes (i.e., we assume *private channels*). A detailed description of the model is given in Section 4.2.

Our protocol (see Section 4.5) is efficient, lightweight, and scalable, since the overhead for maintaining the topology is only polylogarithmic in n: it requires only polylog(n) bits to be processed and sent by each (good) node per round and any node's computation cost per round is also small (polylog(n)). Our protocol guarantees that, despite a large continual adversarial churn and the presence of a large number of Byzantine nodes, with high probability, there is a *large subgraph*[6] present in the network in every round that guarantees efficient communication between *most*[7] good nodes. More specifically, our protocol maintains at least $n - o(n)$ nodes in a structured (hypercube-based), addressed network that is addressable and hence can be used to build a secure, robust, and efficient *DHT* (see Section 4.7.2).

The protocol assumes a *short* (polylog(n) rounds) initial "bootstrap" phase, where there is no churn but has a large number of Byzantine nodes. Without a bootstrap phase, i.e., if there is a large churn from the start, it is easy to show that the adversary can

[6] of size $\geq n - o(n)$

[7] $\geq (1 - o(1))$-fraction

partition the network into large pieces, with no chance of forming even a connected graph. In particular, we assume that in the bootstrap phase the network is a sparse expander which is necessary to tolerate a large number of Byzantine nodes; indeed, all prior works (even) on static networks [39, 64, 67, 70, 122, 72], have assumed expander graphs to tolerate a large number of Byzantine nodes. After the short bootstrap phase, the adversary is free to exercise its power to churn in/out nodes up to the churn limit including adding/removing Byzantine nodes (up to the limit of $\frac{n}{\text{polylog}(n)}$).

Improvement over previous results. To the best of our knowledge, this is the first-known, fully-distributed protocol that guarantees the maintenance of an efficiently routable DHT network under highly dynamic adversarial setting and in the presence of a large number of Byzantine nodes. Our protocol improves several prior results. In particular, it improves over the work of Augustine et al. [9] which guaranteed construction and maintenance of a large ($(n-o(n))$-sized) expander subgraph in the network under adversarial churn *only*, but *not* with Byzantine nodes. Furthermore, the work of [9] does not construct an efficiently routable DHT. The protocol serves as a building block for enabling algorithms for fundamental distributed computing problems such as search and storage, agreement, and leader election in dynamic Byzantine P2P networks. In particular, our protocol can be used to improve over the prior Byzantine protocols for Byzantine agreement [7] and Byzantine leader election [8] and storage and search in dynamic networks [5], which only operated under the assumption that there is an ever-present underlying expander, unlike our current protocol. Our DHT (see Section 4.7.2) guarantees storage and search in dynamic networks even in the presence of Byzantine nodes, unlike prior protocols (e.g., [5]). Another significant improvement is that our protocol can tolerate

both $\frac{n}{\text{polylog}(n)}$ churn and Byzantine nodes, while the above mentioned prior works on agreement [7] and leader election [8] can tolerate only $o(\sqrt{n})$ Byzantine nodes.[8]

4.1.2 Other Related Work and Comparison

There has been a lot of work on P2P protocols for maintaining desirable properties (such as connectivity, low diameter, high expansion, bounded degree) under churn (see e.g., [101, 61, 79, 9] and the references therein), but these do not work under large continual adversarial churn and in the presence of a large number of Byzantine nodes. Most prior algorithms (e.g., [82, 60, 103, 102, 13, 88]) will only work under the assumption that the network will eventually stabilize and stop changing or there is a "repair" time for maintenance when there are no further changes (till the repair/maintenance is finished); these algorithms do not work under high continual adversarial churn (even if there are no Byzantine nodes). There has also been significant prior work in designing P2P networks that are provably robust to a large number of Byzantine faults (e.g., see [43, 56, 96, 111, 14]). These focus on robustly enabling storage and retrieval of data items under adversarial nodes. However, these algorithms will not work in a highly dynamic setting with large, continual, adversarial churn.

The work of [52] is similar in spirit to ours. It also addresses the problem of Byzantine computation in a dynamic setting. It presents a solution for maintaining a clustering of the network where each cluster is $O(\log N)$ size and has more than two thirds honest nodes with high probability. The size of the network, N, can vary polynomially over

[8] However, we note that the works of [7, 8] assume the full information model, in the sense that private channels are not assumed in those works as in this chapter.

time; however the churn is quite small — limited to polylog(N) per round. However, an important contrast to our work is that it does not address the problem of building an efficient and reliable routing structure or DHT. Another main difference compared to our work is that *global knowledge* of the network is assumed during the initialization phase; gaining this knowledge is costly and requires a lot of rounds and messages in a model where only small-sized messages are allowed per edge per round. Also, unlike our work, it assumes that the protocol's actions with respect to previous joins and leaves are over before new joins and leaves happen. Similar to our work, the work of [52] assumes private channels and Byzantine nodes. Some of the approaches used are quite similar to ours, in particular, using *random walks* and maintaining the proportion of honest nodes in a cluster by *shuffling*. However, the technical details are quite different; in particular, since we only assume local knowledge and small-sized messages, our challenge is harder; we use the technical *Byzantine sampling theorem* (see Theorem 7) — which can be of independent interest — to show that random walks have good sampling properties despite the presence of Byzantine nodes. Also, while clustering into $O(\log n)$ nodes is an important first step for our protocol (these are the *committees*), constructing and maintaining an addressable network (hypercube) requires significantly more work.

Kuhn et al. consider in [79] that up to $O(\log n)$ nodes (adversarially chosen) can crash or join per constant number of time steps. Under this amount of churn, it is shown in [79] how to maintain a low peer degree and bounded network diameter in P2P systems by using the hypercube and pancake topologies. There has been lot of work on dynamic network models where *only the edges* change over time but the nodes remain *fixed*, see e.g., [78, 77]; these works do not apply to the churn setting considered here.

Scheideler and Schmid show in [112] how to maintain a distributed heap that allows join and leave operations and, in addition, is resistent to Sybil attacks. A robust distributed implementation of a distributed hash table (DHT) in a P2P network is given by [14], which can withstand two important kinds of attack: adaptive join-leave attacks and adaptive insert/lookup attacks by up to ϵn adversarial peers. This paper assumes that the good nodes always stay in the system and the adversarial nodes are churned out and in, but the *algorithm* determines where to insert the new nodes. Naor and Weider [96] describe a simple DHT scheme that is robust under the following simple random deletion model — each node can fail independently with probability p. They show that their scheme can guarantee logarithmic degree, search time, and message complexity if p is sufficiently small. Hildrum and Kubiatowicz [56] describe how to modify two popular DHTs, Pastry [109] and Tapestry [128] to tolerate random deletions. Several DHT schemes (e.g., [116, 108, 63]) have been shown to be robust under the simple random deletion model mentioned above. There has also been some work on designing fault-tolerant storage systems in a dynamic setting using quorums (e.g., see [97]). However, these do not apply to our model of continual churn. A practical (networking systems) approach to routing under byzantine failures in networks was discussed in [107].

4.2 Computing Model and Problem Definition

4.2.1 An Overview of the Adversarial Network Model

Our network model is based on the model of [9] with the significant difference that our model also incorporates Byzantine nodes. Our goal is to design a protocol that maintains

an overlay network despite adversarial churn and the presence of Byzantine nodes. We consider a synchronous dynamic network controlled by an adversary that is oblivious to the execution of the protocol. The adversary is responsible for designing a very basic dynamic network

$$\mathscr{H} = (H_1, H_2, H_3, \ldots)$$

in an online manner; here, each $H_i = (V_i, E_i')$ is the adversarial network provided at the start of round i. Both the adversary and the protocol can churn out nodes, each up to $\chi(n)$ per round, where $\chi(n)$ is defined as

$$\chi(n) \stackrel{\text{def}}{=} \frac{n}{\log^{\kappa} n}, \tag{4.1}$$

where κ is a fixed constant ≥ 5.[9] The adversary is responsible for bringing in new nodes — equal in number[10] to the number of churned out nodes — and connecting them to existing nodes in a limited fashion (explained below).

Our protocol is required to build an overlay network

$$\mathscr{G} = (G_1, G_2, G_3, \ldots)$$

that guarantees well-connectedness and routability properties (in fact, a *hypercube*) that are essential for building a DHT.

[9] We don't try to optimize the value of κ, but its value depends on the other black box algorithms used in the analysis, e.g., the common coin protocol — see Section 4.4.3 and Byzantine agreement — see Section 4.4.2.

[10] Again, as mentioned in Section 4.1.1, our protocol will also work if the total number of nodes is kept relatively stable, i.e., $\Theta(n)$.

Additionally, the network can contain Byzantine nodes that can collude with each other. The adversary designates a subset $B_i \subset H_i$, $i \geq 1$, to be Byzantine. Once a node is designated to be Byzantine, it remains Byzantine until it is churned out. However, the number of Byzantine nodes $|B_i|$, $i \geq 1$ must not exceed

$$\beta(n) \stackrel{\text{def}}{=} \frac{n}{\log^{\ell} n}, \tag{4.2}$$

where ℓ is a fixed constant ≥ 12.

During the current round r, the Byzantine nodes have full information about both H_i and G_i for all $i \leq r$. Byzantine nodes have unbounded computational power, can behave arbitrarily and maliciously, and can collude with each other. However, they are oblivious to the future behaviors of the adversary and the protocol, so they are unaware of H_i and G_i for $i > r$. As is the usual convention in these settings, the Byzantine nodes are not allowed to lie about their IDs. In other words, when a node (good or Byzantine) sends a message to another node, the recipient will know the correct ID of the sender. We assume *private channels* of communication between nodes, so any message sent between good nodes is unavailable to Byzantine nodes.

During the current round r, the adversary knows a full history of $\mathscr{H}$ until the previous round (i.e., H_{i-1}) including the sets of nodes churned out by the protocol in every prior round. However, it does not know $\mathscr{G}$. The adversary must fix the graph sequence $\mathscr{H} = (H_i; i \geq 1)$ in the following manner.

During the bootstrap phase (when $1 \leq i \leq T_{\mathbf{boot}} = \Theta(\text{polylog}\, n)$). To prepare the protocol to handle adversarial churn and Byzantine nodes, we assume a short initial *bootstrap phase* of $\text{polylog}\,(n)$ rounds. In the bootstrap phase, there is no churn (but

there are Byzantine nodes).[11] At the start of round $i = 1$, the adversary fixes the initial graph $H_1 = (V_1, E_1')$ and designates a subset B_1 of at most $\beta(n)$ (defined in Equation 4.2) nodes as Byzantine nodes (this is sometimes referred to as *oblivious* setting, where the Byzantine nodes are fixed in advance before the start of the protocol). Each node in V_1 has a unique ID chosen from a suitably large (polynomial in n) sized ID space.

Moreover, H_1 is a d-regular expander graph[12] for a fixed d and edge expansion at least a fixed constant $\alpha > 0$. Since the bootstrap phase models a period when the protocol gears up, we require stability. Thus, $H_i = H_{i+1}$ and $B_i = B_{i+1}$, for all $1 \leq i < T_{\text{boot}}$.

During the maintenance phase (when $i > T_{\text{boot}}$). For each round $i > T_{\text{boot}}$, i.e., during the maintenance phase, the adversary first decides which set $C_{i-1}^{\mathsf{adv}} \subset V_{i-1}$ of at most $\chi(n)$ nodes to be churned out at the start of round i. Let $C_{i-1}^{\mathsf{alg}} \subset V_{i-1}$ be the set of nodes that decided to churn themselves out during round $i-1$ in accordance to the protocol; the protocol must ensure that

$$|C_{i-1}^{\mathsf{alg}}| \leq \chi(n).$$

The adversary brings in a new set C_i^{new} of $|(C_{i-1}^{\mathsf{adv}} \cup C_{i-1}^{\mathsf{alg}})|$ nodes,[13] each with a unique ID chosen from a suitably large (polynomial in n) sized ID space. The new vertex set is

[11] As mentioned in Section 4.1.1, without such a phase, the churn adversary can easily partition the network into small disconnected pieces.

[12] As mentioned in Section 4.1.1, an expander graph is needed to tolerate a large number of Byzantine nodes in sparse networks; indeed, all prior works on Byzantine problems in static networks (e.g., [39, 64, 67, 70, 122, 72]) have assumed expander topologies. Without sufficient expansion, Byzantine nodes can be placed so that the graph is effectively partitioned into various parts with no hope of reliable communication between the parts.

[13] The adversary has to keep the network size stable; hence it brings in an equal amount of nodes that are

simply

$$V_i = (V_{i-1} \setminus (C_i^{\mathsf{adv}} \cup C_{i-1}^{\mathsf{alg}})) \cup C_i^{\mathsf{new}}.$$

The adversary then decides E_i' such that each node in C_i^{new} is connected to at least one node in $V_i \setminus C_i^{\mathsf{new}}$. Moreover, no node can have a degree in H_i that exceeds $O(1)$. The adversary must make its choices, essentially its choice of C_i^{adv} and E_i'. The adversary is equipped with the knowledge of $(H_1, \ldots, H_{i-1})$ and C_{i-1}^{alg} while making the aforementioned choices. Note that it is oblivious to the random choices of the protocol.

Our protocol dynamically adds/deletes edges over time. Each $G_i = (V_i, E_i)$ has the same vertex set as that of H_i, while the edges E_i (separate from E_i', the edges of H_i) are determined by the actions of the adversary and the protocol. During each round i, the protocol can add a set $E_i^{\downarrow}$ of new edges. Also, a set $E_i^{\uparrow}$ of edges that were added by the protocol in some previous round is lost — some explicitly removed by the protocol and others removed implicitly because vertices to which they were incident were churned out. Thus, each

$$E_i = (E_{i-1} \setminus E_i^{\uparrow}) \cup E_i^{\downarrow};$$

E_1 is empty. We emphasize that the graph H_i is only presented at round i and the protocol is unaware of future graphs.

The Byzantine nodes have the power to initiate arbitrary number of connections to good nodes; this "flooding" can potentially disrupt the number of connections that good

not churned out. One way to interpret this assumption is that the nodes churned out by the protocol have to be reconnected again by the adversary (they can have the same IDs as well). Also, as mentioned earlier, the protocol will also work fine even if the network size is only approximately n, say within a constant factor.

nodes can accept to other good nodes. However, our protocol has the ability to handle this behaviour of the Byzantine nodes. We assume that, although there is no limit on the number of connections that Byzantine nodes can initiate, the number of connections any Byzantine node can accept is bounded by[14]

$$\Delta = O(\log^3 n) \tag{4.3}$$

Of course, good nodes also have this restriction.

4.2.2 Communication Model

Communication is via message passing. If a node u knows the ID of another node v either directly because u and v are neighbours in the current graph in $\mathscr{G}$, or because they are neighbors in H_i (assuming we are in round i), or indirectly through received messages, then u can communicate with v[15]. Each node can send and receive messages of size polylogarithmic (in n) bits. Furthermore, each good node is restricted to some $M = \Theta(\Delta)$ number of messages it can send and receive (and therefore, the number of nodes to which it can send and receive) in a single round. Note that however, there is no limit on the

[14]This can be considered a "hardware" restriction on all nodes. In any case, without such a restriction, the Byzantine nodes in the dynamic setting (i.e., in the maintenance phase) could potentially create a large hypercube by providing an essentially unlimited number of connections to good nodes. In the dynamic setting, since nodes join and leave, it becomes difficult for good nodes to resist this behavior of the Byzantine nodes. However, we note that this restriction is not needed in the bootstrap phase when the network is static.

[15]This is a typical assumption in the context of P2P and overlay networks, where a node can establish communication with another node if it knows the other node's IP address.

number of messages that Byzantine nodes can send or receive (however, as stated earlier, they can only accept $O(\Delta)$ connections).

4.2.3 Adding/Deleting Edges in $\mathscr{G}$

We assume the existence of a simple two-step handshake protocol (described in the next paragraph) that allows any node u to propose an edge between itself and another node v (provided u knows v's id); the node v, in turn, can either explicitly accept or ignore (thereby implicitly rejecting) the proposal. Furthermore, a node u can drop any edge (u,v) incident to it without consulting v; we assume that v will be immediately notified that (u,v) has been dropped.

4.2.4 Sequence of Events in a Round

We now describe the precise sequence of events that comprise a single round i in our model:

1. The adversary presents the new graph H_i in which some of the old nodes have been churned out and some new nodes have been churned in (assuming round i occurs after the bootstrap phase). The graph G_{i-1} loses all edges that were incident to nodes that were churned out.

2. Each (good) node is then allowed to perform some local computation including private coin flips and send messages[16] to up to $M = \Theta(\Delta)$ other nodes. This

[16]Each message can be of size at most polylog (n).

communication step, among other purposes, is particularly useful for nodes to send requests for adding edges.

3. Each (good) node can again perform some more local computation, and again send messages of size polylogarithmic in n to up to M other nodes. The messages sent in this step are typically responses to messages sent out in the previous step. For instance, a node v that had received a request from a node u to form an edge could either send an accept message or ignore it (implicitly rejecting the request).

4. Once v accepts node u's edge request, edge (u,v) is added to the communication graph. The communication graph now includes all the newly added edges, and is designated G_i. We note that a round is comprised of a two-step protocol to add an edge [9]: the requesting node sends a message to the other node, which takes one more step to accept/reject. The above assumption is slightly different from the classic notion of a round, where messages are sent only one way per round.

4.2.5 Goal

Our goal is to design a distributed protocol to maintain a graph process $\mathscr{G} = (G_1, G_2, \ldots)$ over time such that the following properties are guaranteed with high probability for up to N rounds where N is a polynomial in n number of rounds (say n^c, for some constant $c \geq 1$), where n is the stable network size.

1. **Sparse network:** Each node in G_i should have polylogarithmic degree.

2. **Efficient and reliable communication:** At every round $T_{\text{boot}} < i \leq N$, there must exist a sufficiently large set of nodes $\mathsf{Core}_i \in G_i$ with more than $n - \frac{n}{\text{polylog}(n)}$ nodes

such that every pair of nodes in this set must be able to communicate efficiently and reliably, i.e., they should be able to route a message from one node to the other in polylogarithmic number of rounds and by sending polylogarithmic number of messages.

3. **Distributed Hash Table (DHT):** The network should be addressable and a DHT can be built and maintained on the network, i.e., data items can be stored and efficiently retrieved with only polylogarithmic overhead with high success probability.

4. **Construction and maintenance overhead:** The protocol overhead should be only polylogarithmic in n — only $\text{polylog}(n)$ bits are to be processed and sent by each node per round and any node's computation cost per round is small (polylogarithmic in n).

4.3 Technical Challenges and a High-level Overview of the Protocol

Technical Challenges. The main technical challenge lies in constructing a sparse — even *static* — peer-to-peer network, especially a network where most pairs of honest nodes can route between them efficiently and reliably in the presence of a large number of Byzantine nodes.

Most prior works on Byzantine protocols on sparse networks *assume* an underlying *expander* graph, where the expansion properties prove crucial in solving fundamental problems such as agreement and leader election, see e.g., [39, 122, 72]. However, these

protocols do not guarantee efficient and reliable routing, i.e., routing using polylogarithmic number of hops. The protocol of [72] builds an underlying communication mechanism where messages can be relayed with only polylogarithmic overhead, but the protocol fails to work in a dynamic network. The issue with all the above protocols, as mentioned in Section 4.1 is that they assume that nodes have global knowledge of the network topology to begin with. Such an assumption does not work if one wants to build a dynamic network where the topology keeps changing, or even a static network, where nodes start with local knowledge of only themselves and their immediate neighbors.

The main idea is the following (see Algorithm 5, Algorithm 6, and Section 4.7). Assuming only an underlying *static* sparse expander graph (which is necessary to tolerate a large number of Byzantine nodes in sparse networks as is typical in other Byzantine protocols [39, 122, 72]) where nodes start with only local knowledge (however, they know that the underlying graph is an expander, but nothing about the global topology, except a constant factor estimate of the network size n), we show how to design a routing mechanism so that most honest nodes can communicate efficiently and reliably with each other despite the presence of a large number of Byzantine nodes. In other words, we show how to build an *addressable* network under a Byzantine setting.

The core idea of the protocol is to build an addressable *hypercube network* on top of the underlying static expander network. This part of the protocol is called the "Bootstrap phase" (see Section 4.6) where the network is assumed to be static (i.e., with no churn). The Bootstrap phase is short (polylogarithmic rounds); at the end of this phase the protocol guarantees the construction of a hypercube of large size (i.e., of size $\geq n - o(n)$).

In the subsequent "Maintenance phase" (see Section 4.7), where churn kicks in, the

protocol makes use of the hypercube to guarantee (whp) efficient and reliable communication despite churn and Byzantine nodes. This guarantee holds for a polynomial number of rounds. Once efficient and reliable communication is available, it is straightforward to build a DHT.

Much of the technical challenge lies in the construction of a hypercube in the Bootstrap phase. This is non-trivial as we only have an underlying (arbitrary) expander and nodes have only local knowledge. Note that this network is not addressable. Thus although the diameter is small ($O(\log n)$), it is not clear how to route using only a polylogarithmic number of messages. One can do flooding, but this is inefficient in terms of the number of messages. To the best of our knowledge prior works have not solved this problem.

Our work builds on prior works, in particular, random walks [5, 7, 8], common coin protocol [92] and Byzantine agreement protocols in complete networks [34]. It also introduces several new ideas to build a reliable, robust, and efficient addressable network in a dynamic Byzantine setting that have not been done before.

A High-level Overview of the Protocol. We now describe the main ideas in the construction of the hypercube in the Bootstrap phase.

Our protocol builds the hypercube bottom up, starting from dimension zero. The protocol operates in phases; in phase k, hypercubes of dimension k are built. In phase 0, we construct *cliques* of size $\Theta(\log n)$ each. These cliques act as the building blocks of the hypercube, i.e., the cliques themselves are the nodes of the hypercube. For inductive purposes, the cliques are the zero-dimensional hypercubes. We call these $\Theta(\log n)$-sized cliques *committees* or *supernodes*.

The committees are formed by using *random walks* which is a key technical tool that we use. The basic idea is simple: use random walks to sample tokens (approximately) uniformly at random. All nodes generate tokens (which contain the source node's id) and send those tokens via random walks continually over time. These random walks, once they "mix" (i.e., reach close to the stationary distribution), reach essentially "random" destinations in the network. Thus the destination nodes receive a steady stream of tokens from essentially random nodes, thereby allowing them to sample nodes uniformly from the network. While this is easy to establish in a fully honest network, it is no longer true in a network with Byzantine nodes — these can cause many random walks to be lost and also introduce bias.

We show a technical result called the *Byzantine sampling theorem* (see Section 4.4.1 and Section 4.A) guarantees that the random tokens reach the nodes almost uniformly at random, despite the presence of Byzantine nodes. The Byzantine sampling theorem (which can be of independent interest) is an improvement over a similar result shown in [7, 8] which tolerates only $o(\sqrt{n})$ Byzantine nodes and churn. We note that the work of Guerraoui et al. [52] also uses random walks (see Section 4.1.2 for a comparison.)

In phase 0, $\Theta(\frac{n}{\log n})$ nodes elect themselves as committee leaders and initiate random walks by sending $\Theta(\log n)$ tokens in the underlying expander network. Nodes that get tokens try to join the committees led by the respective leaders. We further show that the Byzantine nodes will never be able to control any more than a $o(1)$ fraction of the total number of committees.

A crucial definition is that of a *good committee*: it is one where at least 90% of the nodes are honest. A good committee can do efficient Byzantine agreement among its

$\Theta(\log n)$ constituent nodes. In fact, we crucially use the fact that they can generate an *unbiased fair coin* (see Section 4.4.3) using a protocol of Micali and Rabin [92]. However, this requires private channels (this is the only place we need this assumption). Unbiased fair coins are used in random shuffling (explained below).

At the end of phase $\texttt{dim}$, the guarantee is that most hypercubes are of dimension $\texttt{dim}$ and they are *good*. A good hypercube is one with *all* of its committees being good (see Definition 14 for the exact definition of a good hypercube). The protocol shows how to construct $(\texttt{dim}+1)$-dimensional hypercubes from $\texttt{dim}$-dimensional hypercubes. We explained above how dimension zero hypercubes are constructed. Since there are only $\frac{n}{\text{polylog}(n)}$ Byzantine nodes, most dimension 0 hypercubes are good. We maintain the invariant in each phase.

A phase consists of four main procedures (see Algorithm 6): *dismantling hypercubes*, *expanding committees*, *shuffling committees*, and *splitting committees*. We give a high-level overview of each.

- **Dismantling:** At the beginning of phase $\texttt{dim}$, about half of the dimension $(\texttt{dim}-1)$ hypercubes dismantle themselves (see Section 4.6.2), i.e., the associated hypercube is destroyed and the nodes become individual nodes. (One has to be careful in not dismantling too many good hypercubes.)

- **Expanding:** The non-dismantled hypercubes will send invitations to the dismantled nodes to join their respective committees. This approximately doubles the size of the committees — this is the "expanding committees" procedure (see Section 4.6.3).

- **Shuffling:** Then the expanded committees shuffle (see Section 4.6.4) the nodes

that joined them by routing them to random committees in their hypercubes — this is necessary so that the Byzantine nodes do not take over a particular committee. This crucially ensures that a good committee remains good.

- **Splitting:** After shuffling, the committees are split (see Section 4.6.6) into two approximately equal parts and then the hypercube of the next higher dimension is created.

There are significant technical challenges in implementing all the above procedures, as Byzantine nodes can behave *arbitrarily* and try to foil each procedure.

A key ingredient of our protocol is to show that while most good hypercubes will double in size and be good again, a bad hypercube does not grow too much in size. This is ensured by the *dissolving* procedure (see Section 4.6.5) which checks whether a hypercube of large enough size is bad.

The main guarantee at the end of the Bootstrap phase is that there will be one large good hypercube of size $n - o(n)$ (see Theorem 9). Since all the committees in this large hypercube are good, routing can now be done efficiently using, say, bit-fixing routing in the hypercube.

In the maintenance phase (see Section 4.7) we show that despite large adversarial churn and the presence of Byzantine nodes, the large good hypercube will be maintained with high probability for a polynomial number of rounds. There are a couple of crucial algorithmic techniques that we use to ensure maintenance. If the adversary is able to predict the composition of the committees, it can churn out whole committees. So to overcome this, the incoming new nodes added by the adversary to some node in the large hypercube are routed to random committees. Moreover, this random location is

maintained as an invariant throughout the life of all nodes via frequent shuffling of nodes – both good and bad – to random locations within the hypercube. This ensures that every committee has a sufficiently large $\Theta(\log n)$-sized members list. Additionally, since the Byzantine nodes are also shuffled to random locations, they cannot congregate in any one or few select committees. Rather, we can show that no committee has more than some small percentage of bad nodes (whp).

There is one crucial issue that crops up if we limit ourselves to random shuffling of nodes. The churn can slowly remove nodes from the large hypercube and incoming nodes can be connected to nodes that are not in the large good hypercube, thereby depleting nodes from it. To overcome this situation, we build some verification procedures — some straightforward (like nodes ensuring that they are in a committee), while others are not. The most powerful verification we perform is committees establishing connections to random other committees through a verified bit-fixing sequence (see Algorithm 11). Consider some structure (hypercube or otherwise) other than the large good hypercube built by the adversary. The good committees will seek to form these verified connections with randomly chosen committees in the hypercube. These verified connections are essentially formation of pairwise edges between nodes in either committees. But, due to the degree restriction of the Byzantine nodes, they will be unable to provide sufficient number of edges required for verification. Thus, verification will fail for a lot of good committees and the structure will dissolve and request reconnections (see Theorem 10). There will not be sufficient nodes in the structures disconnected from the good large hypercube, so incoming nodes will be forced to join the large hypercube, thereby ensuring that the number of nodes in the large good hypercube is not depleted.

4.4 Preliminaries

4.4.1 Sampling in a Static Network with Byzantine Nodes

We describe and analyze a mechanism to achieve an *almost uniform, random sampling* of nodes in a static network with Byzantine nodes. Its use is limited to the bootstrap phase because we require the underlying network to be a static expander graph. We closely emulate the works of [7] and [8] and modify them suitably to serve our purpose. More precisely, we execute the code in Algorithm 4.

Remark 6. The length of the random walks τ is defined as $\tau \stackrel{\text{def}}{=} M \log n$, where M is a large enough (but fixed) positive constant.

Emulating the works of [7] and [8], we can show that

Theorem 7. *Let $T_{walk} = \tau(\tau+1) = \Theta(\log^2 n)$ and consider an n-node (static) expander network G under an oblivious (Byzantine) adversary, and suppose that at most $\beta(n)$ nodes are Byzantine, where $\beta(n) \stackrel{\text{def}}{=} \frac{n}{\log^{\ell} n}$. Also suppose that each of the honest nodes introduces $c_2 \log^2 n$ random walk tokens at time zero (for some suitable positive constant c_2) and those random walks are executed according to the mechanism described previously.*

Then there exists a set of honest nodes `Core` *of size $\geq n - O(\frac{n}{\log^{\ell} n})$ such that, in every time interval $[iT_{walk}+1, (i+1)T_{walk}]$ for $0 \leq i \leq \Theta(\log n)$, the following hold:*

1. *A random walk token originating from a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to terminate at any particular node in* `Core`*.*

2. *A random walk token that terminates at a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to have originated at any particular node in* `Core`*.*

3. *At most $O(\frac{n}{\log^{\ell-4} n})$ nodes in* `Core` *receive tokens that did not originate in* `Core` *or did not take all their steps among nodes in* `Core`*.*

We defer the detailed proof of this theorem to Section 4.A.

Definition 9. *Let $|$*`Core`*$|$ be equal to $n-\gamma_1(n)$. Then from the above theorem, $\gamma_1(n) = O(\beta(n)) = O(\frac{n}{\log^{\ell} n})$.*

Definition 10. *Let* `Inner` *(called the "inner-core") be the set of nodes in* `Core` *that receive only those tokens that:*

1. *originated in* `Core`*, and*

2. *took all their steps in* `Core`*.*

It follows from Theorem 7 that

Observation 10. $|$`Inner`$| \geq n - O(\frac{n}{\log^{\ell-4} n})$.

Definition 11. *Let $|$*`Inner`*$|$ be equal to $n-\gamma_2(n)$. Then from the above observation, $\gamma_2(n) = O(\frac{n}{\log^{\ell-4} n})$.*

4.4.2 Majority Consensus Protocol Under Byzantine nodes

Our protocol uses Byzantine agreement in complete networks (of $O(\log n)$ size, i.e., in the committees) extensively as a basic subroutine. More specifically, our subroutine solves majority consensus where the agreed upon (common) value should be the majority (input) value among all nodes. In particular, in our applications, we want to solve the majority consensus problem when almost all the good nodes (say, $\geq 80\%$ of all nodes) have the same input value.

We use the Byzantine agreement protocol of Dolev et al. [34] which is deterministic and solves the everywhere Byzantine agreement in complete networks using $O(Nt + t^3 \log t)$ message bits, where N is the number of the nodes in the network and $t \leq N(\frac{1}{3} - \varepsilon)$ is the number of Byzantine nodes, for any positive constant ε. The running time of this algorithm is $O(t)$ rounds. We will use this algorithm extensively for solving Byzantine agreement in a good committee, i.e., the number of Byzantine nodes (see Definition 13) is at most $1/10$ fraction of the total number of nodes. Note that in this setting, Dolev's agreement will also yield majority agreement if more than (say) 80% of all nodes have the same input value. Also since the committee size is $O(\log n)$, the message and time bounds of Dolev et al.'s algorithm is polylogarithmic in n.

We also note that this algorithm works not just for binary consensus (i.e., two input values), but also for multi-valued inputs, as long as most of the good nodes (say, $\geq 80\%$ of all nodes) have the common input value.

4.4.3 Unbiased Common Coin Protocol

Our algorithm requires generation of an unbiased (fair) common coin among a clique of $\Theta(\log n)$ nodes connected to each other (these are the nodes of a supernode, i.e., a committee). We assume that the clique is dominated by good nodes, i.e., less than a $(\frac{1}{10})$-fraction of the nodes are Byzantine. This clique is referred to as *good committee* in our protocol. To get an unbiased common coin among the good nodes we use the protocol of Micali and Rabin [92] that requires (only) private channels between every pair of good nodes. This protocol can be used to generate an unbiased common coin, i.e., the probability of getting a 0 or 1 is exactly $\frac{1}{2}$. The message complexity of this protocol

is polynomial in size of the set, and since we apply this protocol to a set of size $O(\log n)$, each node sends at most polylog (n) messages when executing this protocol.

We note that the common coin protocol of Micali and Rabin requires an expected constant number of rounds to finish and has zero probability of error. The protocol can have a message size of polylog (n) bits when executed on a set of size $O(\log n)$ and the constant expected time is under this assumption. Thus, even allowing only $O(\log n)$ bits per round per edge the round complexity is expected polylog (n). It is easy to modify this protocol to terminate in polylog (n) rounds with high probability — this is needed for our algorithm for synchronization purposes (nodes need to know a high probability bound on the termination time to proceed to the next step of the algorithm).

This modification is as follows. Run the common coin protocol in phases. Each phase takes at most polylog (n) rounds, say $\eta \log^2 n$ rounds, for some large (but fixed) constant η. Two things can happen at the end of a phase for any node: either the common coin protocol finishes for that node or it doesn't. To make all nodes agree whether the protocol finished successfully for all nodes or not, we run a Byzantine agreement algorithm due to Dolev et al. [34]. The Dolev et al. agreement algorithm is *deterministic* and finishes in polylog (n) rounds since the clique has at most $O(\log n)$ Byzantine nodes and the message sizes are also polylog (n). At the end, all nodes will agree whether the common coin protocol finished or not. If they agree on the former, then the common coin value output by the common coin protocol will be taken as the common coin; otherwise all (good) nodes repeat the common coin protocol again (i.e., start another phase). Since the success probability of the common coin protocol is at least a constant in a phase, the protocol will succeed with high probability after $O(\log n)$ phases. Hence we have the

following theorem.

Theorem 8. *Given a clique of* $\Theta(\log n)$ *nodes there is a protocol that generates an unbiased common coin with probability at least* $1 - \frac{1}{n^3}$ *in* $\text{polylog}(n)$ *rounds. The protocol has* $\text{polylog}(n)$ *message complexity.*

By the above theorem, we can assume that all common coin flips executed by all the committee nodes over the course of the protocol for a polynomial number of rounds succeed with high probability (via a union bound).

4.5 The Algorithm

The structure of the algorithm follows the structure of the adversarial network model presented in Section 4.2.1, i.e., it consists of a *bootstrap phase* and a *maintenance phase*. During the bootstrap phase (see Section 4.6) the goal is to assemble the nodes into committees and then organize these committees to form a *hypercube* of large size, i.e., $n - o(n)$. Note that in the (short) bootstrap phase, there is no churn but there are up to $\beta(n) = \frac{n}{\text{polylog}(n)}$ Byzantine nodes. The resilience and self-repairing capability of the constructed hypercube is exploited during the maintenance phase (see Section 4.7) to make the network resistant against churn and influences of the Byzantine nodes. See Algorithm 5 for a formal (but high-level) description along with references to more detailed and rigorous expositions of individual steps.

4.6 The Bootstrap Phase

A hypercube of dimension k, say, is built up in a bottom-up fashion. This phase begins with cliques of size $\Theta(\log n)$ each. Those cliques act as the building blocks of the hypercube, i.e., the cliques themselves are the nodes of the hypercube. In other words, the cliques themselves can be viewed as zero-dimensional hypercubes.

We obey the following convention. We call a committee *good* if at least 90% of its members (nodes) are good. We call a hypercube good if *all* of its nodes (that is, committees) are good, and moreover, no more than one-fifteenth fraction of its (individual) nodes are Byzantine.

Lemma 30 and Lemma 35 tell us that at the end of Phase 0, most [17] committees are good committees. We can rephrase that as *at the beginning of Phase* 1*, most hypercubes, each of which is of dimension zero, are good.*

Thus this is the *invariant* that we would like to maintain at the end of phase k — *most hypercubes, each of which is of dimension k, are good.*

4.6.1 Forming Committees

Remark 7. In the execution of Line 8 in Algorithm 7, there is one subtle mechanism that is crucial in preventing the Byzantine nodes from forming new committees or invading existing committees willy-nilly: When a recipient node v accepts the invitation from a sender node w, v lets w know about its acceptance decision through the same random walk path by which w's invitation had reached v in the first place. In other words, only the edges of the underlying expander H_1 are used in the committee formation process.

[17] i.e., $\geq 1 - o(1)$ fraction

No overlay edges are used. This limits the communication capability — and thus the influence — of the Byzantine nodes (see Lemma 35).

Lemma 30. *If $L_{\texttt{Core}}$ is a random variable that denotes the number of committee leaders in* `Core`*, then with high probability,*

$$\frac{(1-\delta_1)(n-\gamma_1(n))}{c_1 \log n} < L_{\texttt{Core}} < \frac{(1+\delta_1)(n-\gamma_1(n))}{c_1 \log n},$$

where δ_1 is any arbitrarily small (but fixed) positive constant.

Proof. Let X_v be an indicator random variable that denotes if an honest node v becomes a committee leader. Then $E[X_v] = \Pr[X_v = 1] = \frac{1}{c_1 \log n}$.

Thus

$$L_{\texttt{Core}} = \sum_{v \in \texttt{Core}} X_v.$$

Then, by linearity of expectation,

$$E[L_{\texttt{Core}}] = \sum_{v \in \texttt{Core}} E[X_v] = (n - \gamma_1(n)) \cdot \frac{1}{c_1 \log n} = \frac{n - \gamma_1(n)}{c_1 \log n}.$$

By the Chernoff bound [94, Theorem 4.4],

$$\begin{aligned} &\Pr[L_{\texttt{Core}} \geq (1+\delta_1) E[L_{\texttt{Core}}]] \leq \exp(-\frac{\delta_1^2 \cdot E[L_{\texttt{Core}}]}{3}) \\ &\implies \Pr[L_{\texttt{Core}} \geq \frac{(1+\delta_1)(n-\gamma_1(n))}{c_1 \log n}] \leq \exp(-\frac{\delta_1^2 (n-\gamma_1(n))}{3 c_1 \log n}). \end{aligned}$$

Similarly by [94, Theorem 4.5],

$$\begin{aligned} &\Pr[X_L \leq (1-\delta_1) E[L_{\texttt{Core}}]] \leq \exp(-\frac{\delta_1^2 \cdot E[L_{\texttt{Core}}]}{2}) \\ &\implies \Pr[L_{\texttt{Core}} \leq \frac{(1-\delta_1)(n-\gamma_1(n))}{c_1 \log n}] \leq \exp(-\frac{\delta_1^2 (n-\gamma_1(n))}{2 c_1 \log n}). \end{aligned}$$

Thus,

$$\Pr[\frac{(1-\delta_1)(n-\gamma_1(n))}{c_1 \log n} < L_{\texttt{Core}} < \frac{(1+\delta_1)(n-\gamma_1(n))}{c_1 \log n}]$$
$$\geq 1 - (\exp(-\frac{\delta_1^2(n-\gamma_1(n))}{3c_1 \log n}) + \exp(-\frac{\delta_1^2(n-\gamma_1(n))}{2c_1 \log n}))$$
$$\geq 1 - \frac{1}{\Omega(n^c)}, \text{ for any positive constant } c.$$

□

Definition 12. *For any non-empty subset U of V, let $\mathscr{T}(U)$ denote the set of tokens originating from U.*

It follows from the protocol that

Observation 11. $|\mathscr{T}(\texttt{Core})| = L_{\texttt{Core}} \cdot c_2 \log^2 n$.

We recall that $\mathscr{I}_v$ is the set of invitations that reach v after T rounds, i.e., at the end of the random walks. The following lemma states that $|\mathscr{I}_v|$ is concentrated around $\frac{c_2}{c_1} \log n$ with high probability.

Lemma 31. *For $v \in$ `Inner`, it holds with high probability that $(1-\delta_2)\frac{c_2}{c_1} \log n < |\mathscr{I}_v| < (1+\delta_2)\frac{c_2}{c_1} \log n$, where δ_2 is any arbitrarily small (but fixed) positive constant.*

Proof. Let v be any node in `Core` and let t be any token originating in the core. Let $Y_{v,t}$ be an indicator random variable that denotes if token t ends up at node v after the random walks are finished. Then by Theorem 7,

$$E[Y_{v,t}] = \Pr[Y_{v,t}] \implies \tfrac{1}{n} - \tfrac{1}{n^5} \leq E[Y_{v,t}] \leq \tfrac{1}{n} + \tfrac{1}{n^5}.$$

Let Y_v be a random variable that denotes the number of tokens originating in $\texttt{Core}$ that end up at node v after the random walks are finished. Then

$$Y_v = \sum_{t \in \mathscr{T}(\texttt{Core})} Y_{v,t}.$$

By linearity of expectation,

$$E[Y_v] = \sum_{t \in \mathscr{T}(\texttt{Core})} E[Y_{v,t}] = |\mathscr{T}(\texttt{Core})| \cdot E[Y_{v,t}].$$

But $|\mathscr{T}(\texttt{Core})| = L_{\texttt{Core}} \cdot c_2 \log^2 n$ by virtue of Observation 11. Thus

$$E[Y_v] = L_{\texttt{Core}} \cdot c_2 \log^2 n \cdot E[Y_{v,t}].$$

From Lemma 30 it follows that, with high probability,

$$\tfrac{(1-\delta_1)(n-\gamma_1(n))}{c_1 \log n} \cdot c_2 \log^2 n \cdot (\tfrac{1}{n} - \tfrac{1}{n^5}) < E[Y_v] < \tfrac{(1+\delta_1)(n-\gamma_1(n))}{c_1 \log n} \cdot c_2 \log^2 n \cdot (\tfrac{1}{n} + \tfrac{1}{n^5}).$$

Simplifying, it holds with high probability that

$$(1-\delta_1)\tfrac{c_2}{c_1} \log n \cdot (n - \gamma_1(n))(\tfrac{1}{n} - \tfrac{1}{n^5}) < E[Y_v] < (1+\delta_1)\tfrac{c_2}{c_1} \log n \cdot (n - \gamma_1(n))(\tfrac{1}{n} + \tfrac{1}{n^5}).$$

Simplifying again, we get (with high probability),

$$(1-\delta_4)\tfrac{c_2}{c_1} \log n < E[Y_v] < (1+\delta_4)\tfrac{c_2}{c_1} \log n,$$

where $0 < \delta_4 < 1$ is a constant. It should be noted that δ_4 is a function of δ_1. But since δ_1 can be made arbitrarily small, δ_4 can be made arbitrarily small too.

Now if we restrict v to be in $\texttt{Inner}$, then that would imply that v receives no tokens from outside the $\texttt{Core}$. Thus, for $v \in \texttt{Inner}$, $Y_v = |\mathscr{I}_v|$. Therefore,

$$(1-\delta_4)\tfrac{c_2}{c_1} \log n < E[|\mathscr{I}_v|] < (1+\delta_4)\tfrac{c_2}{c_1} \log n,$$

where δ_4 is any arbitrarily small (but fixed) positive constant.

By the Chernoff bound [94, Theorem 4.4], for any fixed positive constant δ_5,

$$\begin{aligned}
&\Pr[|\mathscr{I}_v| \geq (1+\delta_5)E[|\mathscr{I}_v|]] \leq \exp(-\frac{\delta_5^2 \cdot E[|\mathscr{I}_v|]}{3}) \\
&< \exp(-\frac{\delta_5^2 \cdot (1-\delta_4)\frac{c_2}{c_1}\log n}{3}) < n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{3c_1}} \\
&\implies \Pr[|\mathscr{I}_v| \geq (1+\delta_4)(1+\delta_5)\frac{c_2}{c_1}\log n] < n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{3c_1}}.
\end{aligned}$$

Similarly, by applying the Chernoff bound [94, Theorem 4.5] again,

$$\begin{aligned}
&\Pr[|\mathscr{I}_v| \leq (1-\delta_5)E[|\mathscr{I}_v|]] \leq \exp(-\frac{\delta_5^2 \cdot E[|\mathscr{I}_v|]}{2}) \\
&< \exp(-\frac{\delta_5^2 \cdot (1-\delta_4)\frac{c_2}{c_1}\log n}{2}) < n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{2c_1}} \\
&\implies \Pr[|\mathscr{I}_v| \leq (1-\delta_4)(1-\delta_5)\frac{c_2}{c_1}\log n] < n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{2c_1}}.
\end{aligned}$$

Thus

$$\begin{aligned}
&\Pr[(1-\delta_4)(1-\delta_5)\frac{c_2}{c_1}\log n < |\mathscr{I}_v| < (1+\delta_4)(1+\delta_5)\frac{c_2}{c_1}\log n] \\
&> 1 - (n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{3c_1}} + n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{2c_1}}) \\
&> 1 - 2n^{-\delta_5^2(1-\delta_4)\cdot\frac{c_2}{3c_1}}.
\end{aligned}$$

We can choose the constants so that $\delta_5^2(1-\delta_4)\cdot\frac{c_2}{3c_1} > 5$. Thus

$$\Pr[(1-\delta_4)(1-\delta_5)\tfrac{c_2}{c_1}\log n < |\mathscr{I}_v| < (1+\delta_4)(1+\delta_5)\tfrac{c_2}{c_1}\log n] > 1 - \tfrac{2}{n^5}.$$

As both δ_4 and δ_5 can be made arbitrarily small, we can further simplify and get

$$\Pr[(1-\delta_2)\tfrac{c_2}{c_1}\log n < |\mathscr{I}_v| < (1+\delta_2)\tfrac{c_2}{c_1}\log n] > 1 - \tfrac{2}{n^5},$$

where δ_2 is any arbitrarily small (but fixed) positive constant. □

Lemma 32. *For a committee leader $v \in \texttt{Inner}$, let C_v be a random variable that denotes the number of nodes that decide to join the committee led by v. Then, with high probability,*

$$(1-\delta_3)c_1 \log n < C_v < (1+\delta_3)c_1 \log n,$$

where δ_3 is any arbitrarily small (but fixed) positive constant.

Proof. For any particular node $v \in \texttt{Core}$, v is equally likely to join any of the $L_{\texttt{Core}}$ leaders. Thus on expectation, every leader will have $\frac{|\texttt{Core}|}{L_{\texttt{Core}}}$ members in his committee. Applying the result of Lemma 30, $\frac{c_1 \log n}{1+\delta_1} < E[C_v] < \frac{c_1 \log n}{1-\delta_1}$ with high probability. Applying a standard Chernoff bound, we have the lemma. □

Finally, we observe that the committee-formation protocol (see Algorithm 7) essentially takes twice the length (duration) of the random sampling process (see Algorithm 4), which takes $O(\log^2 n)$ rounds. Thus

Lemma 33 (Time complexity of the committee-formation phase (Algorithm 7))**.** *The committee-formation phase (see Algorithm 7) takes $O(\log^2 n)$ rounds.*

Committees and Hypercubes — Some Definitions and Some Size-bounds.

Definition 13. *We say that a committee is "good" if the number of Byzantine nodes in the committee is at most a one-tenth fraction of the total number of nodes in the committee. A committee is called "bad" if it is not good.*

Extending this to hypercubes, we say that

Definition 14. *A hypercube is said to be "good" if* every *committee in it is good, and furthermore, the total number of Byzantine nodes in the hypercube is at most* $(\frac{1}{15})$*-th fraction of the total number of nodes (Byzantine or otherwise) in it.*

A hypercube is called "bad" if it is not good.

Observation 12. During the later phases of the algorithm, when the hypercubes are "large enough", any hypercube with size $\omega(\beta(n))$ is good if and only if all its committees are good.

Proof. This is simply because of the fact that the total number of Byzantine nodes in the network is $O(\beta(n))$, thus they cannot constitute a constant fraction of a hypercube with $\omega(\beta(n))$ nodes. □

Observation 13. During the earlier phases of the algorithm, when the hypercubes are still "small enough" (e.g., of size $O(\beta(n))$), at most $(\frac{1}{\text{polylog}(n)})$-fraction of them will be "bad" inspite of all their committees being "good".

Proof. As the total number of Byzantine nodes in the network is limited by $\beta(n) = \frac{n}{\text{polylog}(n)}$, only (at most) $(\frac{1}{\text{polylog}(n)})$-fraction of all the hypercubes can have a constant fraction of their nodes be Byzantine. □

Next we show that the Byzantine nodes cannot corrupt too many committees during the committee-formation protocol (see Algorithm 7), because they cannot even reach (by sending messages) too many good nodes in a small (polylog (n) rounds, say) number of rounds.

Definition 15. *We say that a node v is "affected" if v is honest and it is a neighbor of some Byzantine node. An honest node that is not "affected" is called "unaffected".*

Each Byzantine node is connected to at most d honest nodes. Thus the number of affected nodes is at most $d\beta(n)$. Thus if we denote the set of unaffected nodes by `Unaffected`, then

Observation 14. $|\texttt{Unaffected}| \geq n - (d+1)\beta(n)$.

Using similar arguments as used in the proof of Theorem 12 in Section 4.A, we can show that at most $\beta(n) \cdot \log^4 n$ *unaffected* nodes receive tokens generated by the Byzantine nodes. For the sake of completeness, we lay down the details of the proof here.

Lemma 34. *During the course of the "committee formation protocol" (see Algorithm 7)*[18]*, at most $O(\beta(n) \cdot \log^4 n)$ nodes in `Unaffected` receive tokens that did not originate in `Unaffected` or did not take all their steps among nodes in `Unaffected`.*

Proof. We bound the number of tokens that are received by nodes in `Unaffected` in any particular interval $[iT_{\text{walk}} + 1, (i+1)T_{\text{walk}}]$: Recall that nodes in `Core` are never adjacent to Byzantine nodes. Thus, any token that is sent across a link of the cut $(\texttt{Unaffected}, V \setminus \texttt{Unaffected})$ was sent by an honest node. It follows from Lemma 49 that (whp) at most $2h\log^2 n \cdot T_{\text{walk}} = 2h\log^2 n \cdot \tau(\tau+1) = 2h\log^2 n \cdot M\log n(M\log n + 1) = O(\log^4 n)$ tokens are sent across any edge during $[iT_{\text{walk}}, (i+1)T_{\text{walk}}]$.

The size of the cut $(\texttt{Unaffected}, V \setminus \texttt{Unaffected})$ is bounded by $d(n|Unaffected|)$. Therefore, at most $O(\log^4 n) \cdot d(n|Core|) = O(\beta(n) \cdot \log^4 n)$ tokens originating from outside `Unaffected` reach nodes in `Unaffected` and vice versa. Thus at most $O(\beta(n) \cdot \log^4 n)$ nodes in `Unaffected` receive tokens that did not originate in `Unaffected` or did not take all their steps in `Core`. □

[18] which takes $O(\log^2 n)$ rounds, as stated by Lemma 33

An immediate consequence of this is that the number of committeess that the Byzantine nodes can dominate — or even be a part of — is limited by the same quantity $O(\beta(n) \cdot \log^4 n)$. Thus we have

Lemma 35. *After the committee formation phase, there can be at most $O(\beta(n) \cdot \log^4 n)$ bad committees in the network. The number of good nodes inadvertently stuck in those bad committees is limited by the same quantity $O(\beta(n) \cdot \log^4 n)$ as well.*

4.6.2 Dismantling

- **Assumption:** We have some k *good* hypercubes, each of dimension `dim`. We assume that committees can flip unbiased coins as needed (up to polylog(n) per round per committee). Note that it is easy to ensure that `dim` is common knowledge among all nodes.

- **Goal:** At the end, k^* hypercubes remain intact, i.e., some $(k-k^*) \in [(1-\delta_6)\frac{k}{2}, (1+\delta_6)\frac{k}{2}]$ hypercubes must decide to dismantle themselves, where δ_6 is any arbitrarily small (but fixed) positive constant. The nodes in these hypercubes become free. We also want to ensure that this process terminates in a clean manner with exactly one remnant hypercube.

We perform this step in one of two ways depending on whether $k \geq c_3 \log n$ (for some sufficiently large constant c_3) or otherwise.

4.6.2.1 Case 1: $k \geq c_3 \log n$.

At a high level, each hypercube `Hyp` tosses an unbiased coin and decides to dismantle if and only if the outcome is 1. To implement this high level idea, the committee with all its bits 0 (zero) tosses a coin (using the unbiased common coin protocol — see Section 4.4.3) and broadcasts the result (dismantle or not) to all other nodes (i.e., committees) in the hypercube. Of course, when a committee hears the "dismantle" verdict, all good nodes within the committee simply delete all incident overlay edges.

Remark 8. A Byzantine node present in a good committee may try to keep its adjacent overlay edges, but the good nodes present at the other end of those edges will follow the protocol and duly delete those edges. Of course there may be edges where both the incident nodes are Byzantine, but we do not care about those edges since those do not affect any good node.

Lemma 36. *For any arbitrarily small (but fixed) $\delta_6 > 0$, there exists a suitably large constant c_3 such that, as long as the number of hypercubes $k \geq c_3 \log n$, after the above steps to dismantle half the hypercubes are executed, we will whp be left with $k' \in [(1-\delta_6)\frac{k}{2}, (1+\delta_6)\frac{k}{2}]$ intact hypercubes.*

Proof. Let X be the number of hypercubes that are not dismantled. Clearly, $\mathrm{E}[X] = \frac{k}{2}$ and since the hypercubes decide whether to dismantle or not independent of each other,

we can apply Chernoff bounds and get

$$\begin{aligned}
&\Pr[|X - \mathrm{E}[X]| \geq \delta_6 k] \\
&\leq 2\exp(-\frac{k{\delta_6}^2}{6}) \\
&\leq 2\exp(-\frac{c_3{\delta_6}^2 \log n}{6}) && \text{(since } k \geq c_3 \log n\text{, by assumption)} \\
&< 2\exp(-5\log n) && \text{(since } c_3 > \tfrac{30}{{\delta_6}^2}\text{)} \\
&< \frac{2}{n^5}.
\end{aligned}$$

□

4.6.2.2 Case 2: $k < c_3 \log n$.

We proceed as before, where each hypercube decides to dismantle itself with probability $\frac{1}{2}$. However, since the total number of hypercubes is now small, the expected number of hypercubes to dismantle is also small; therefore, we can no longer apply Chernoff bound techniques as before.

Instead, we argue that even when we cannot exactly predict the number of dismantling hypercubes, we can still achieve the desired result — that of building higher-dimensional hypercube(s) — albeit in a slightly different fashion.

So when each of the k hypercubes decides to dismantle itself (independently) with probability $\frac{1}{2}$, one of the following three things can happen:

1. **Case 1: All of the hypercubes decide to dismantle:** We argue below that this is the only "bad" case; we denote this event by `Failure`. Then $\Pr[\texttt{Failure}] = 2^{-k}$, which can be as large as $\frac{1}{4}$ (in the boundary case where only two hypercubes are

left, and both of them decide to dismantle). But even in that worst possible case, Pr[Failure] is still a constant, i.e., it can safely be said that Pr[Failure] = $O(1)$.

Thus, if only there were some *detection mechanism* that would detect that the algorithm has experienced Failure, then we would be able to just *repeat* this *Bernoulli trial*, and arrive at our desired outcome — with high probability — in $O(\log n)$ repetitions.[19]

The detection mechanism. We now show that such a detection mechanism actually exists. During the "expansion" phase (see Section 4.6.3 and Algorithm 8 in particular), every good, intact committee (i.e., a committee in a good hypercube that did not dismantle) sends out invitations to the newly dismantled nodes. So if there were no intact hypercubes to begin with — which the case would be were Failure to occur, then no such invitations would be sent, and that silence — the complete lack of invitations — would serve as a signal to all the nodes in the network (at least to all the nodes in Core; see Theorem 7) that Failure has indeed happened.

There is one subtle issue here though that we must address: the Byzantine nodes may send out (false) invitations even when there are no intact hypercubes (i.e., when Failure has occurred). But they would be able to reach only $\frac{n}{\text{polylog}(n)} =$

[19]We call such a desired outcome Success, which denotes the event that at most $(k-1)$-hypercubes decide to dismantle. We note that until Success happens, the actual dismantling process does not take place. Each individual iteration of the Bernoulli trial concenrs only the decision-making process — the decision of whether or not to dismantle.

$o(n)$ nodes (see Theorem 7); thus the majority of nodes (i.e., $\geq n - o(n)$ nodes) would know the truth, namely, `Failure` has happened, and they will move on to the next iteration of the Bernoulli trial.

2. **Case 2: Fewer than $\lceil \frac{k}{2} \rceil$ hypercubes decide to dismantle:** This does not pose a problem for us, except that we allow the dismantling process to continue a little longer (see Remark 9 and Remarks 11). That is, the hypercubes that have already decided to dismantle no longer takes part in the Bernoulli trial; only the hypercubes that have decided to remain intact participate in the Bernoulli trial again. This process goes on until at least $\lceil \frac{k}{2} \rceil$ hypercubes decide to dismantle (totalling over all the iterations).

3. **Case 3: At least $\lceil \frac{k}{2} \rceil$ hypercubes decide to dismantle:** This is the ideal scenario for us; if/when this happens, the (current iteration of) the *dismantling phase* is completed, and the algorithm moves on to the *expansion phase* (see Section 4.6.3).

Remark 9. For the committees in the remaining (intact) hypercubes to expand properly (see Section 4.6.3), it is necessary that the number of dismantled committees (i.e., committees belonging to hypercubes that dismantled) be at least as much as the number of remaining committees (i.e., committees in hypercubes that did *not* dismantle). Please refer to the explanation in Remark 11.

To achieve that, we make the dismantling process to go on for a little longer in **Case** 2 above.

To summarize, we have the following result:

Lemma 37. *If there were k hypercubes at the beginning of the dismantling phase (for some positive integer k), then at the end of the dismantling phase, there would be k^* hypercubes remaining, where k^* is also a positive integer such that $k^* \leq \lfloor \frac{k}{2} \rfloor$.*

4.6.3 Expanding

4.6.3.1 Case 1: $k \geq c_3 \log n$.

- **Assumption:** There are $n_c \in [(1-\delta_8)\frac{n-\gamma_2(n)}{2}, (1+\delta_8)\frac{n-\gamma_2(n)}{2}]$ committee nodes and $n_f \in [(1-\delta_8)\frac{n-\gamma_2(n)}{2}, (1+\delta_8)\frac{n-\gamma_2(n)}{2}]$ free (newly dismantled) nodes in `Inner`. It holds that $n_c + n_f = |\texttt{Inner}| = n - \gamma_2(n)$.

- **Goal:** All the committees in `Inner` roughly double in size, i.e., they grow up to a size in $[(1-\delta_9)2c_1 \log n, (1+\delta_9)2c_1 \log n]$.

Remark 10. As is the case in Algorithm 7, there is one subtle mechanism here that is crucial in preventing the Byzantine nodes from forming new committees or invading existing committees willy-nilly: When a recipient node v accepts the invitation from a sender node w, v lets w know about its acceptance decision through the same random walk path by which w's invitation had reached v in the first place. In other words, only the edges of the underlying expander H_1 are used in the committee formation process. No overlay edges are used. This limits the communication capability — and thus the influence — of the Byzantine nodes (see Lemma 35).

Lemma 38. *Let $H \in$ `Inner` be any good hypercube that was not dismantled in the dismantling phase (see Section 4.6.2). Let C be any committe in H. Then after the*

execution of the expansion procedure (i.e., Procedure 8 as described above), it will hold with high probability that

$$|C| \in [(1-\delta_9)2c_1 \log n, (1+\delta_9)2c_1 \log n],$$

where δ_9 is any arbitrarily small (but fixed) positive constant.

Proof sketch. The free nodes will receive invitations from (almost) random committees; thus this expansion process can be thought of as a "balls-into-bins" problem. Performing analysis very similar to the one done in Section 4.6.1 (Lemmas 31 and 32 in particular), we have the result. □

4.6.3.2 Case 2: $k < c_3 \log n$.

- **Assumption:** There are k^* good hypercubes, where $k^* \leq \lfloor \frac{k}{2} \rfloor$, the same quantity as defined in Lemma 37. Each of these hypercubes are composed of committees of size $\approx c_1 \log n$ (see Lemma 32).
- **Goal:** Each committee in the remaining k^* good hypercubes increases in size to $\approx (\frac{k}{k^*}) \times c_1 \log n$.

We proceed similarly as before (for the case when k was large) and achieve the above-mentioned goal in a similar manner to that of Lemma 38. More precisely, we show the following:

Lemma 39. *Let $H \in$ `Inner` be any good hypercube that was not dismantled in the dismantling phase (see Section 4.6.2). Let C be any committee in H. Then after the execution of the expansion procedure (i.e., Procedure 8 as described above), it will hold with high probability that*

$$|C| \in [(1-\delta_{10})\frac{kc_1 \log n}{k^*}, (1+\delta_{10})\frac{kc_1 \log n}{k^*}],$$

where δ_{10} is any arbitrarily small (but fixed) positive constant.

Remark 11. We here explain in more detail why we need at least $\lceil \frac{k}{2} \rceil$ hypercubes to dismantle when $k < c_3 \log n$ (see Case 2 and Case 3 in Section 4.6.2):

Suppose we remove this restriction; then in the worst case, k^* can be as large as $k-1$, in which case only one hypercube (comprising $\Theta(\frac{n}{\log^2 n})$ committees, i.e., $\Theta(\frac{n}{\log n})$ individual nodes) gets dismantled. Thus the number of "free" nodes is $\Theta(\frac{n}{\log n})$, which are to be distributed/shared among $\Theta(\frac{n}{\log n})$ "intact" committees. Thinking of this as a "balls into bins" problem, the expected number of (new) balls that a bin will get is $\Theta(1)$, and therefore we cannot apply Chernoff bound to show the desired result.

To circumvent this problem, we define k^*, which denotes the number of remaining hypercubes, in such a way (see Lemma 37) that the number of dismantled committees is always as large as the number of intact committees. This ensures that the number of dismantled (individual) nodes is always a log-factor higher than the number of remaining committees, i.e., in the "balls-into-bins" problem mentioned in the preceding paragraph, the number of "balls" is always a log-factor higher than the number of "bins". Thus the expected number of new balls that a bin well get is now $\Theta(\log n)$, and this allows us to apply a Chernoff bound and show the desired result.

Remark 12. Lemma 39 tells us that after the execution of this *expansion phase* (i.e., that described and analyzed in Section 4.6.3.2), the committees in the good hypercubes may more than double in size. This is not a problem because the *splitting phase* (see Section 4.6.6.2) takes care of this.

In fact, we argue that such a "slack" in the permissible range of (good) committee

sizes is even *necessary* to handle the fact that our goal is to end up with a giant[20] hypercube — the hypercube is a structure that contains exactly $2^{\texttt{dim}^*}$ coordinate positions (for some positive number $\texttt{dim}^*$), whereas the total number of nodes in the network, i.e., n, need *not* be an exact power of 2 (in fact, n need not contain any kind of special properties whatsoever).

4.6.4 Shuffling

Goal. The goal of the shuffling procedure presented in this section is to randomize the composition of the committees within a good hypercube. More precisely, after the algorithm terminates, each node — whether it was already part of the hypercube or was newly recruited by the algorithm presented in the previous section— has to be assigned to a random committee of the hypercube.

As in the previous section, we only consider good hypercubes (see Definition 14), where all committees are good..

The Shuffling Protocol. The shuffling protocol works in a good hypercube. It is assumed that all (good) nodes in the hypercube know the dimension of the hypercube (which depends on the current phase number).

Unlike the previous procedures — forming committees, dismantling and expanding – which use the underlying expander graph, the shuffling protocol uses the hypercube. In this respect, it is essentially the same as the shuffling procedure that is done repeatedly in the *maintenance phase* to maintain the large hypercube (of size $\geq n - o(n)$) that is constructed at the end of the bootstrap phase. Accordingly, to avoid repetition, we will

[20] comprising $\geq n - o(n)$ nodes

defer the details and analysis of the shuffling procedure to the maintenance phase (see Section 4.7), and give only a high-level overview as required in the bootstrap phase.

The main idea behind the shuffling protocol is as follows. It moves all nodes around to random locations in polylog (n) rounds. Each committee as a whole is responsible for relocating its nodes to random locations. For every node in a committee, say C, its random (committee) location in the hypercube is determined by using the common coin protocol. Then the node is "routed" to its destination committee by using a procedure called "verified bit-fixing". The route is the usual bit-fixing route in the hypercube, but in order to avoid the actions of the Byzantine nodes, the node is "chaperoned" from the source committee to the destination committee. Each step in the bit-fixing route is done by running Byzantine agreement between successive committees on the route. This ensures that all nodes, good and bad, follow the bit fixing route correctly. The details are spelled out in Algorithm 12. This shuffling algorithm invokes an important subroutine whenever a node has to be handed over from one committee to another. Such handovers are always to verified random committees obtained via a recently completed execution of Algorithm 11. So handovers can take effect immediately. The handover details are described in Algorithm 13.

The verified bit-fixing uses several subprotocols. One is a basic protocol called **Intra-Committee Verification** that is executed by every node u, whereby u ensures that it is in a committee C. For u to be fooled into thinking that it is in a valid committee, the Byzantine nodes must invest at least $\Omega(\log n)$ edges to ensure u's illusion that it belongs to a committee. Recall that each Byzantine node has a degree bound of $\Delta = \text{polylog}\,(n)$. This immediately ensures that at most $O(\frac{\Delta \cdot \beta(n)}{\log n})$ nodes can be fooled in this manner. This

protocol is described in detail in Algorithm 9.

When committee C obtains a connection request from a random committee C', it is guaranteed two things. Firstly, C' has been chosen uniformly and independently at random from the set of all committees in the hypercube. Secondly, both C and C' know that the bit fixing path from C to C' was traversed in a verified manner, thereby allowing them to communicate and handover nodes without further checks. C obtains such a verified random committee C' by choosing a random location within the hypercube and bit-fixing to that location in a manner whereby each step of the bit-fixing is through verified connections obtained via Byzantine agreement (see Algorithm 9 for this **Inter-Committee Verified Connection** procedure). At the end of this bit-fixing, C gets a verified connection to C' which is uniformly random. The details are in Algorithm 11.

Using arguments similar to those in Section 4.7 the following lemma can be shown.

Lemma 40. *The Shuffling protocol guarantees (whp) that each committee of the newly-formed (shuffled) hypercube contains at most a* $(\frac{1}{12})$*-fraction of Byzantine nodes. The number of rounds is* $\text{polylog}(n)$ *and each node sends and receives at most polylogarithmic messages per round.*

4.6.5 Dissolving

The main goal of this section is to show that no bad hypercube will grow beyond $\frac{n}{\text{polylog}(n)}$ in size. The dissolving procedure is used by good nodes to detect whether they are in a bad hypercube and if so, they will leave the hypercube.

The dissolving procedure is again very similar to the ideas used in the shuffling procedure (cf. Section 4.6.4) and is described in detail in the maintenance phase (see

Section 4.7). The point is that the same dissolving procedure is used in the maintenance phase as well to make sure that a bad hypercube does not grow too much in size. The same applies to the bootstrap phase as well, and in fact is a bit simpler, since there are no node joins and leaves as in the maintenance phase. We give a brief idea of the dissolving procedure and defer the details to Section 4.7.

The dissolving procedure can be considered as an extension of the shuffling procedure. The basic idea is simple. Consider a bad hypercube that is of large size, i.e., $\omega(\beta(n)$, where $\beta(n)$ is the number of Byzantine nodes. There will be a large number of good committees (in fact, most of the committees will be good) in such a hypercube. Each good committee will faithfully execute the shuffling procedure to its nodes move to random committee locations. As described in the shuffling procedure, the verified bit-fixing protocol (Algorithm 11) ensures that either the nodes move to their correct destination or if not (due to bad behaviour of intermediate or destination committees which are bad, or presence of "holes" due to dismantling of good committees), they will be detected via the procedures, **Intra-Committee Verification**, **Inter-Committee Verified Connection** and Algorithm 13. Once a good committee detects that the shuffling has failed, then it will dismantle all its nodes. This dismantling creates "holes" in the hypercube which leads to further dismantling.

Algorithm 11 forces committees to be constantly connecting to random other committees. This will happen in a load balanced manner in a good hypercube. However, when the hypercube is not good, these random bit fixings will lead to locations that cannot be handled by Byzantine nodes due to their limited degree budget. Algorithm 9 for this **Inter-Committee Verified Connection** and Algorithm 9 for **Intra-Committee**

Verification come with several checks to ensure that all good nodes are able to relocate to random committees. We refer to Section 4.7 for a detailed description and analysis.

Lemma 41. *The dissolving procedure ensures that no bad hypercube will grow beyond* $\frac{n}{\log^5 n}$ *in size.*

4.6.6 Splitting

- **Assumption:** All nodes are organized in committees in hypercubes of dimension $\texttt{dim}$.
- **Goal:** Split each committee to form a hypercube of dimension $\texttt{dim}+1$.

4.6.6.1 Case 1: $k \geq c_3 \log n$.

Let us consider a good hypercube. Lemma 38 tells us that each committee in the current hypercube has size $\Theta(\log n)$, which is approximately double the size compared to what the committees had started out with at the beginning of Section 4.6.4. We recall that the hypercubes grow (dimension-wise) in synchrony. For the sake of analysis, let us assume that we are at the point in the algorithm where the good hypercubes are of dimension $\texttt{dim}$.

In this section, we describe and analyze an algorithm to restore the committee sizes to their original size (i.e., approximately half the current size). The algorithm does that by *splitting* each committee into two (approximately) equally-sized committees and then joining the comittees with a "super-edge" (details given below) to grow the $(\texttt{dim}+1)^{\text{th}}$ dimension of the hypercube.

Algorithm for Splitting.

1. Let C be a committee in the hypercube. We recall that C is "good", i.e., at least 90% of the nodes in C are honest. Using the common coin protocol of Section 4.4.3, C picks a hash function h, say, uniformly at random from a universal family of hash functions $\mathscr{U}$, say, where $\mathscr{U}$ is hardcoded into all the nodes. What h does is to map $ID(v)$ to 0 or 1 — uniformly at random. This new bit acts as the bit value for the coordinate of v in the $(\texttt{dim}+1)^{\text{th}}$ dimension of the hypercube.

2. We recall that it follows from the committee formation process in Section 4.6.4 that all the nodes in a committee C are aware of the IDs of all the other nodes. Thus, once h is fixed, all the nodes in C automatically know about the new $(\texttt{dim}+1)^{\text{th}}$ dimension coordinates of all the other nodes in C. Thus if C is to be split into two committees C_{left} and C_{right}, say, then every node in C knows not only its own place (whether it's in C_{left} or in C_{right}), but also the placements of all the other nodes in C.

   Thus there can happen a *consistent* splitting of C into C_{left} and C_{right}. We observe that the clique structure on C is the same as the complete bipartite graph structure on $C_{\text{left}} \cup C_{\text{right}}$. Therefore no new edges need to be formed within/across C_{left} and C_{right}.

3. For each neighboring committee C' of C in the old, d-dimensional hypercube, all the nodes in C inform all the nodes in C' about the composition of C_{left} and C_{right}, and vice versa. Then all the nodes in C_{left} and C'_{left} coordinate and build a complete bipartite graph structure on $C_{\text{left}} \cup C'_{\text{left}}$, who become neighbors in the

$\texttt{dim}^{\text{th}}$ dimension, say. Similarly, all the nodes in C_{right} and C'_{right} coordinate and build a complete bipartite graph structure on $C_{\text{right}} \cup C'_{\text{right}}$, who become neighbors in the $(\texttt{dim}+1)^{\text{th}}$ dimension.

Remark 13. In Step 3 above, the influence of messages sent by Byzantine nodes can be completely ignored by letting the good nodes only consider messages that were sent by at least a $(\frac{9}{10})$-fraction of the nodes in its committee or a neighboring committee. As in the previous section, we exploit the fact that the majority of the nodes in a committee are good to filter out messages from Byzantine nodes.

Observation 15. Since there are $\Theta(\log n)$ nodes in both C and C', for any two good committees C and C', at most $\Theta(\log^2 n)$ messages are exchanged in the splitting protocol described above. Also, the above splitting protocol takes only $O(\log n)$ rounds.

Next we show that each of the newly formed committees, viz. C_{left} and C_{right}, is of size $\Theta(\log n)$. We show this by showing that C is (almost) equally split during the formation of C_{left} and C_{right}.

Lemma 42. *For any fixed positive constant $0 < \delta < 1$, it holds with high probability that*

$$(1-\delta)\tfrac{|C|}{2} < |C_{left}|, |C_{right}| < (1+\delta)\tfrac{|C|}{2}.$$

Proof. From the property of the hash function h it follows that for any node $v \in C$, $\Pr[v \text{ goes to } C_{\text{left}}] = \Pr[h(v) = 0] = \Pr[v \text{ goes to } C_{\text{right}}] = \Pr[h(v) = 1] = \frac{1}{2}$. Thus the expected number of nodes in C_{left} (or C_{right}) is $\frac{|C|}{2}$. Using standard Chernoff bound arguments, we have the lemma. □

Next we show that both the newly formed committees, viz. C_{left} and C_{right}, are *good* with high probability.

Lemma 43. *It holds whp that for every committee C in the hypercube, the corresponding C_{left} and C_{right} are good committees.*

Proof. Let b be the number of Byzantine nodes in C. Let b_{left} and b_{right} be the number of Byzantine nodes in C_{left} and C_{right} respectively.

We have from Lemma 40 that $b \leq \frac{|C|}{12}$. From the property of the hash function h it follows that for any node $v \in C$, $\Pr[v \text{ goes to } C_{\text{left}}] = \Pr[h(v) = 0] = \Pr[v \text{ goes to } C_{\text{right}}] = \Pr[h(v) = 1] = \frac{1}{2}$. Thus the expected number of Byzantine nodes in C_{left} (or C_{right}) is at most $\frac{|C|}{24}$. Using standard Chernoff bound arguments, we can show that the following holds with high probability:

$$b_{\text{left}} \leq \frac{|C|}{22} < \frac{|C_{\text{left}}|}{10},$$

and

$$b_{\text{right}} \leq \frac{|C|}{22} < \frac{|C_{\text{right}}|}{10}.$$

That is, both C_{left} and C_{right} are *good*.

Taking a union bound over $O(\frac{n}{\log n})$ committees present in the network, we have the lemma. □

Now we are prepared to present the main result of this section.

Lemma 44. *A good hypercube of dimension* `dim` *(where* `dim` *is a non-negative integer) at the beginning of execution of Section 4.6.6 is restructured into a good hypercube of dimension* `dim`$+1$ *by the end of execution of Section 4.6.6. This newly formed hypercube of dimension* `dim`$+1$ *is formed exclusively of good committees, each of size* $\Theta(\log n)$.

4.6.6.2 Case 2: $k < c_3 \log n$.

We recall from Section 4.6.2.2 and Section 4.6.3.2 that the committees may grow up to a factor of $\approx \frac{k}{k^*}$ in this case, i.e., the committees may grow well beyond double their original size.

This, however, does not pose a problem for us as we simply continue the *splitting process* as described before (see Section 4.6.6.1) for a little longer — until the committee sizes come down to the prescribed range, which is between (approximately) $c_1 \log n$ and $2c_1 \log n$ (see Lemma 32, Lemma 38, Lemma 39, and Remark 12).

Note that this specifically means, among other things, that we may "jump" dimensions in doing so, i.e., after the execution of this phase, all the good hypercubes may grow up to a dimension of `dim'` where `dim'` is some natural number that may be greater than `dim` by even 2 or more.[21]

4.6.7 Putting the Pieces Together

Lemma 41 guarantees that no bad hypercube will grow beyond a certain size. Now we argue that there will always (in every phase) be *some* good hypercube, and thus at the end of the bootstrap phase, there will be one good hypercube of the desired size, which is $\geq n - o(n)$.

Theorem 9. *At the end of the bootstrap phase, there will be one good hypercube of size* $\geq n - o(n)$.

[21] Recall that `dim` was the original dimension at the beginning of the *splitting phase* (see Section 4.6.6). Under "normal circumstances", i.e., those analyzed in the previous section (see Section 4.6.6.1), `dim'` would be exactly equal to `dim` + 1.

Proof sketch. In the earlier phases, i.e., as long as the hypercubes are of size $o(\frac{n}{\log^5 n})$, the *number* of good hypercubes k is too large (i.e., $\omega(\beta(n))$), thus by a simple counting argument, the Byzantine nodes will not be able to corrupt every good hypercube. Thus there will always be at least one good hypercube (many, actually) that will continue to grow.

In the latter phases, i.e., when the hypercubes are of size $\omega(\frac{n}{\log^5 n})$, all the hypercubes are good, because no bad hypercube is able to grow up to this size (see Lemma 41). □

4.7 The Maintenance Phase

Once the hypercube is formed during the bootstrap phase, we will need to maintain it for a sufficiently long $\text{poly}(n)$ period of time despite near-linear churn and the presence of Byzantine nodes. There are a few important factors that need to be addressed. Firstly, during the bootstrap phase, we had the advantage of an underlying expander graph that we could use reliably (of course, taking into account the influence of Byzantine nodes). In the maintenance phase however we must rely on the hypercube to provide us with the facility to sample committees uniformly at random. Secondly, we now have to tackle the effects of both the Byzantine nodes as well as the adversary. Thirdly, on the design front, we will need to ensure that there is exactly one large hypercube that we call the *dominant hypercube*. This dominant hypercube comprises $\Theta(\frac{n}{\log n})$ committees as its hypercube vertices, thus it is of dimension $d = \Theta(\log n)$ and the precise value of d is assumed to be common knowledge.[22] The Byzantine nodes may try to create other hypercube(s) or other structures designed to mimic hypercubes. We have to be careful in our design to be

[22]This value of d is built into the protocol in the bootstrap phase.

able to ensure that there will be a single dominant hypercube containing $n - o(n)$ nodes and $\Omega(\frac{n}{\log n})$ committees.

Definition 16. *We say that a committee C is healthy until round $r \geq T_{boot}$ if it was a committee in the dominant hypercube at the end of the bootstrap phase and comprised $\Theta(\log n)$ nodes from T_{boot} until round r such that at most 10% of the nodes were Byzantine at any point in time.*

Extending this to hypercubes, we say

Definition 17. *We call the dominant hypercube healthy until round r if its dimension is d and all of its 2^d committees were healthy until round r.*

Our goal is to design the maintenance phase so that we can ensure the following.

- **Healthy Dominant Hypercube:** Recall that the boostrap phase ensured the creation of a single dominant hypercube of dimension d comprising $2^d \in \Theta(\frac{n}{\log n})$ committees as hypercube coordinates. We are required to ensure that the dominant hypercube survives in a healthy way for a sufficiently large polynomial in n number of rounds.

- **Efficient Integration:** A node that is not a member of some committee in the dominant hypercube is said to be *unintegrated*. (Similarly, committees that are not part of the dominant hypercube are called *unintegrated committees*.) We must ensure that the number of unintegrated nodes at any time is at most $\frac{n}{\text{polylog}(n)}$. Recall that when a node requests reconnection, the adversary is required to reconnect it afresh to a pre-existing node. Our model allows a small number of up to $\frac{n}{\text{polylog}(n)}$

nodes to be repeatedly connected to Byzantine nodes and never integrate despite reconnection requests.

We now present several interacting algorithmic components of the maintenance phase. We start with the very basic protocol **Intra-Committee Verification** executed by every node u, whereby u ensures that it is in a committee C. For u to be fooled into thinking that it is in a valid committee, the Byzantine nodes must invest at least $\Omega(\log n)$ edges to ensure u's illusion that it belongs to a committee. Recall that each Byzantine node has a degree bound of $\Delta = \text{polylog}(n)$. This immediately ensures that at most $O(\frac{\Delta \cdot \beta(n)}{\log n})$ nodes can be fooled in this manner. This protocol is described in detail in Algorithm 9.

Notice that the committees can interact with each other through message passing. We will now describe the mechanics of this message passing so that, in the rest of the section, we can describe the protocol at the level of committees rather than individual nodes. If C_1 and C_2 are two neighboring committees, each is assumed to be aware of the list of members in the other; Any update in membership due to joins/leaves are assumed to be informed to neighbours in every round. If C_1 wants to send a message M to C_2, then, C_1 first performs a local Byzantine agreement to decide on the exact composition of the message M. Then, every member of C_1 sends M to every node in C_2. Then, at C_2, the members must perform a Byzantine agreement to agree that M was indeed received. If C_1 and C_2 are not neighbors, then C_2 must be expecting a message from C_1. Otherwise, nodes in C_2 will not accept messages from C_1. When C_2 is expecting a message from C_1, the procedure to send the message is the same as if they were neighbors.

Our next protocol provides each committee C in a healthy hypercube with a steady supply of connections to verified random committees within its hypercube. When C

obtains such a connection C', it is guaranteed two things. Firstly, C' has been chosen uniformly and independently at random from the set of all committees in the hypercube. Secondly, both C and C' know that the bit fixing path from C to C' was traversed in a verified manner, thereby allowing them to communicate and handover nodes without further checks. C obtains such a verified random committee C' by choosing a random location within the hypercube and bit-fixing to that location in a manner whereby each step of the bit-fixing is through verified connections obtained via Byzantine agreement (see Algorithm 10 for this **Inter-Committee Verified Connection** procedure). At the end of this bit-fixing, C gets a verified connection to C' which is uniformly random. The steady stream of such connections is obtained by running multiple executions of these bit-fixing protocols in parallel and in a pipelined fashion. The details are in Algorithm 11 and the following claim follows.

Lemma 45. *Consider a hypercube of dimension d. If it has remained healthy for the recent* $\text{polylog}(n)$ *rounds, then every committee is guaranteed to receive* $\Theta(\log n)$ *verified connections to committees chosen uniformly at random (with replacement) and independently from the committees that comprise the hypercube.*

We must note an important contribution of Algorithm 11. It forces committees to be constantly connecting to random other committees. This will happen in a load balanced manner within a healthy hypercube. However, when the hypercube is not healthy, these random bit fixings will lead to locations that cannot be handled by Byzantine nodes due to their limited degree budget. Algorithm 9 for this **Inter-Committee Verified Connection** and Algorithm 9 for **Intra-Committee Verification** come with several checks to ensure that good nodes (barring some small $o(n)$ of them) and healthy committees in unhealthy

hypercubes will learn that they are in unhealthy hypercubes and take remedial action in which they request the adversary to reconnect them.

Now that we have established a way to make random verified connections, we must ensure that committees are maintained in a healthy manner despite churn and the effects of Byzantine nodes. We do this by shuffling the nodes around so that — at any time — each node (even Byzantine) in a healthy hypercube is in a random committee within the hypercube. We accomplish this by moving nodes around to random locations every polylog (n) rounds. The details are spelled out in Algorithm 12.

This shuffling algorithm invokes an important subroutine whenever a node has to be handed over from one committee to another. Such handovers are always to verified random committees obtained via a recently completed execution of Algorithm 11. So handovers can take effect immediately. The handover details are described in Algorithm 13. Additionally, this handover protocol comes in handy for integrating new nodes into a hypercube.

When the adversary connects a new node u to a node v in some committee C, node v can initiate Algorithm 14 to get u connected to random committee within the hypercube. Care should be taken to ensure that v cannot influence or know the random location where u will be sent. We achieve this by enforcing a waiting time long enough to ensure that the verified random committee to which u is sent was chosen only after u was put in the list for handover.

4.7.1 Analysis

We recall from Equation 4.2 that ℓ is a fixed constant ≥ 12, i.e., $\beta(n) \leq \frac{n}{\log^{12} n}$. We also recall the definition of Δ from Equation 4.3. The immediate implication is that

Observation 16. The sum of the degrees of Byzantine nodes $D = \Delta\beta(n)$ cannot exceed $O(\frac{n}{\log^9 n})$.

We recall from Equation 4.1 that κ is a fixed constant ≥ 5, thereby ensuring that at most $O(\frac{n}{\log^5 n})$ nodes are churned in/out per round.

Now let us focus our attention on a single hypercube of dimension $\texttt{dim}$ with $2^{\texttt{dim}}$ committees. Suppose further that the hypercube has $\Omega(2^{\texttt{dim}} \cdot \log n)$ nodes already integrated into it. At the start of the maintenance phase, since the nodes (both good and bad) are placed uniformly and independently at random during the bootstrap phase, each committee has $\Theta(\log n)$ nodes out of which at most 10% are bad. Thus, the dominant hypercube starts out healthy and we may assume that it has remained healthy for $\text{polylog}(n)$ rounds when we allow the bootstrap phase to extend for an extra $\text{polylog}(n)$ rounds.

Lemma 46. *Suppose the maintenance phase starts with a single healthy dominant hypercube that has been healthy for the recent* $\text{polylog}(n)$ *rounds with at least* $\Theta(2^{dim} \cdot \log n) = n - o(n)$ *nodes integrated into it. Then, the dominant hypercube remains healthy whp in the current round.*

Proof Sketch. Since each committee gets $\Theta(\log n)$ samples in every round by Lemma 45, there would have been sufficient samples to ensure that every integrated node's current position – especially those slotted for handover in the current round – is chosen randomly, so each committee clearly has $\Theta(\log n)$ nodes (by a simple balls into bins argument).

Notice that the size of the committees can be lower bounded by $c \log n$ for any fixed c by adjusting the constant within the bound of $\Theta(\frac{n}{\log n})$ for the number of committees. Moreover, the (at most) $O(\frac{n}{\log^4 n})$ nodes that were churned in during the last $O(\log n)$ rounds will also be in random locations. Recall that the Byzantine nodes are also chaperoned to random locations, so no node (good or bad) can reach a node other than the one chosen randomly by their (good) committee. Thus, they are also in random locations. So when we ensure a suitable lower bound on the size of the committees, we can use a simple Chernoff bound argument to ensure that no more than 10% of the nodes in some fixed committee is Byzantine with some very high probability and then use the union bound to extend the claim to all committees. □

Lemma 47. *Let U_r denote the set of unintegrated nodes during round r. Then, whp, $|U_r| \in O(\frac{n}{\log^3 n})$ for some polynomial in n rounds.*

Proof. Our argument is largely based on elementary counting. For the sake of clarity, we have not optimized the logarithmic factors.

We first claim that the total number of committees with at least 10% bad nodes is at most $O(\frac{n}{\log^{10} n})$ because each such committee drains at least $\Omega(\log n)$ out of the total degree budget D of Byzantine nodes which we observed earlier was at most $O(\frac{n}{\log^9 n})$.

When U_r reaches a cardinality in $\Theta(\frac{n}{\log^3 n})$, the number of healthy committees that are unintegrated must be $\Theta(\frac{n}{\log^4 n})$ because each node in U_r must be part of some committee (otherwise, it will just ask for reconnection).

However, each such committee C will seek to connect with $\Theta(\log n)$ random committees while attempting to execute Verified-Random-Bit-Fixing(). Each such pair-wise connection between committees will require $\Theta(\log^2 n)$ edges to be formed. Since the

number of good committees is only $\Theta(\frac{n}{\log^4 n})$, at least half the random committees sought by C will be non-existent (whp). Each such connection to a non-existent committee requires $\Theta(\log^2 n)$ edges to be formed between C and the random non-existent committee. Thus, for C to continue without dissolving, the Byzantine nodes must expend $\Theta(\log^3 n)$ out of its degree budget D. Then, thanks to Observation 16, at most $O(\frac{n}{\log^{12} n})$ such committees will remain without dissolving. This means that $\Theta(\frac{n}{\log^4 n})$ committees (comprising $\Theta(\frac{n}{\log^3 n})$ nodes) will dissolve (whp). □

Thus, by the above lemma, whenever the number of unintegrated nodes reaches $\Omega(\frac{n}{\log^3 n})$, all but $O(\frac{n}{\log^3 n})$ nodes dissolve whp. Thus,

Theorem 10. *Suppose at the start of the maintenance phase the dominant hypercube is healthy and the number of nodes that are not in the dominant hypercube is at most* $O(\frac{n}{\log^3 n})$. *Then the dominant hypercube will remain good for* $\text{poly}(n)$ *rounds with high probability. Moreover, whp the number of nodes in non-dominant hypercubes will remain bounded from above by* $O(\frac{n}{\log^3 n})$.

Remark 14. We note that the protocols described in this section are quite general in nature and can be used in the bootstrap phase even for hypercubes of arbitrary dimension in the range $[1, \texttt{dim}]$.

4.7.2 Building a Distributed Hash Table (DHT)

It is easy to show that a DHT can be built on top of the dominant hypercube (i.e., of size at least $n - o(n)$), e.g., see [61]. The DHT can be used to store and retrieve (search) data items efficiently and reliably with low storage overhead despite the presence of Byzantine

nodes and adversarial churn. We use a scheme similar to that of [61]. We give a brief overview here, and refer to [61] for the technical details.

The scheme uses *consistent hashing* (see e.g., [116, 61]) to map data items to hypercube locations (recall that a hypercube location is a hypercube of dimension zero which is made up of a committee of $\Theta(\log n)$ nodes). The data (or key) is hashed to a value using a strongly universal hash function whose range consists of the range of possible hypercube locations (say, between $[0, \frac{n}{2\log n}]$).[23] Data is inserted to a hypercube location whose coordinate is equal to the hash value of the data. The data is replicated and stored in all the $\Theta(\log n)$ committee nodes of the hypercube location. Similarly, when a node joins a committee, it takes over the replicas of data items stored in the committee it joins. Since the dominant hypercube's committees don't get destroyed all of a sudden, the data persists. The routing mechanism of the dominant hypercube allows for efficient search and retrieval of data even under Byzantine nodes and churn. Search for this data item is thus directed to the hypercube location whose coordinate is equal to the data's hashed value. Since there are $\Theta(\log n)$ committee nodes in a location, search will succeed even if only one node in committee is alive in the network (this node will have the data).

4.8 Conclusion

Designing efficient and lightweight distributed computation protocols that are provably robust to the high amount of dynamism (where *both* nodes and edges change continually over time) and also tolerant to presence of Byzantine nodes in a P2P network is an

[23] since the dominant hypercube is guaranteed to contain $n - o(n)$ nodes and each hypercube location is made of $\Theta(\log n)$ (committee) nodes.

important step in realizing the goal of fully-autonomous decentralized computation even in the presence of large number of Byzantine nodes and large-scale dynamism.

We took a step towards the above goal and presented a protocol for construction and maintenance of a dynamic network with good properties even in presence of large number of Byzantine nodes and large continual adversarial churn. The dynamic network guarantees fast and reliable routing and can be used as a DHT. This is the first such protocol that can guarantee these properties under a large number of Byzantine nodes and a large amount of churn.

Our work leaves several questions open. First, our protocol can tolerate only up to $\frac{n}{\text{polylog}(n)}$ Byzantine nodes and churn. An important question is whether these bounds can be improved, in particular, to tolerate a linear fraction of Byzantine nodes and churn; this looks especially challenging in a sparse network and requires possibly a new approach. Second, it will be good to remove the assumption of private channels and design a protocol with similar guarantees in the full information model where Byzantine know about the good nodes' communications as well.

Algorithm 4 Random Sampling Algorithm. Code for node v.

Procedure Random-Sampling

for all (good) nodes u **do**

    Initiate $c_2 \log^2 n$ random walk tokens for some suitable constant c_2.

    Each token is stamped with the source node's ID and a counter initialized to 0.

    ▷ We maintain $M \log n$ FIFO queues pertaining to the different counter values, where M is a (suitably large) fixed positive constant (fixed in analysis).

    for all $0 \leq i < M \log n$ rounds **do**

        Select the oldest (up to) $c_2 \log^2 n$ tokens from the i^{th} FIFO queue.

        Increment their counters.

        Send each selected token to a random neighbor of u in the underlying expander network.

    end for

    Receive tokens and add each to the appropriate FIFO queue if counter value $< M \log n$.

    for all tokens with counter value $M \log n$ **do**

        Harvest the source IDs from the tokens and use as random samples.

        Delete tokens after source IDs have been harvested.

    end for

end for

Algorithm 5 The Fast Byzantine Routing Algorithm in Dynamic Networks

1: **Procedure** Main

2: **Call** Bootstrap. ▷ During this phase, committees are formed and they organize themselves into a hypercube.

3: **Call** Maintenance. ▷ The hypercube structure is maintained despite churn.

Algorithm 6 Algorithm for the *bootstrap phase*

1: **Procedure** Bootstrap

2: **Call** Form-Committees. ▷ See Section 4.6.1. Form $\Theta(\frac{n}{\log n})$ committees each with $\Theta(\log n)$ nodes. Each committee is a hypercube of dimension 0.

3: **repeat**

4: **Call** Dismantle-Hypercubes. ▷ See Section 4.6.2. Each hypercube dismantles with probability $\frac{1}{2}$.

5: ▷ When the number of hypercubes is small, all of them can dismantle, which is unintended.

6: **if** all committees dismantled **then**

7: Undo dismantling and continue to the next iteration of the loop.

8: **end if**

9: **Call** Expand-Committees ▷ See Section 4.6.3. The nodes from dismantled hypercubes join random committees.

10: **Call** Shuffle-Committees ▷ See Section 4.6.4. The nodes in a hypercube randomly shuffle themselves among the committees within the hypercube.

11: **Call** Split-Committees ▷ See Section 4.6.6. Each committee in a (good) hypercube splits itself into two almost equal parts and forms a part of a hypercube of one higher dimension.

12: **until** there is one hypercube of size $\geq n - o(n)$

Algorithm 7 Committee Formation Protocol.

1: **Procedure** Form-Committees

2: **for** every (good) node v **do**

3: v chooses to be a *committee leader* with probability $\frac{1}{c_1 \log n}$, where c_1 is a suitably large constant (e.g., 9 or 10, say).

4: **if** v becomes a committee leader **then**

5: v sends invitations to $c_2 \log^2 n$ random nodes. c_2 is a suitably large constant (e.g., 200 or 250, say). (The random sampling is done by executing Algorithm 4.)

6: **end if**

7: **end for**

8: **Concurrent** to the above **for** loop, every free node waits until it receives (one or more) invitations and accepts one invitation (selected uniformly at random from the set of invitations it has received in this round).

Algorithm 8 Committee Expansion Protocol for large values of k.

Procedure Expand-Committees

for every (good) committee C **do**

Send invitations to $c_2 \log^2 n$ random nodes (the random sampling is done by executing Procedure 4).

if invitation accepted by some node v **then**

Send a message to all nodes in C introducing v.

end if

end for

Concurrent to the above **for** loop, every free node waits until it receives (one or more) invitations and accepts one invitation (selected uniformly at random from the set of invitations it has received in this round).

Algorithm 9 Intra-Committee Verification. This procedure comprises three components (listed and described below) that are continually executed by every node u. We use C to denote u's committee.

1: **Dimension Check.** Let d be the dimension of the Hypercube, which we assume is common knowledge. If that dimension is not maintained (i.e., the coordinates for C is not d bits long in the hypercube), then u immediately requests reconnection.

2: **Byzantine Agreement Checks.** Several steps are of the protocol require invoking Byzantine agreement. When a good node u detects that its committee is not executing the Byzantine agreement protocol faithfully, it will request a reconnection. In particular, if u proposes a bit value b and is aware that more than 80% of the nodes. in the committee proposed b, the final decision must be b. If this is violated, u must request for a reconnection.

3: **Member List Verification.** Node u must ensure that C contains a sufficiently large $\Omega(\log n)$ sized membership. To ensure this, the list of members of C must be common knowledge to all its members and this list is maintained and updated via Byzantine agreement whenever a node joins or leaves.

Algorithm 10 Inter-Committee Verified Connection. This procedure is invoked by a committee C to establish a verified connection with another committee C'. Additionally, C can check if C' is *good*. C' must cooperate only under one of the following conditions: (i) C and C' are neighbors or (ii) some neighbor C'' of C' must be willing to introduce C to C'; otherwise, C' must refuse to cooperate.

1: **if** C and C' are neighbors **then**
2: C must have already established connection with C'' that neighbors C'
3: C requests C'' to introduce it to C' and C'' complies.
4: **end if**
5: Every member of C sends an edge request to every member of C'.
6: Members of C'' comply (when conditions are met).
7: Nodes in C run Byzantine agreement to ensure that at least (9/10)-fraction of C connected with $(9/10)$-fraction of C''.
8: **if** the Byzantine agreement was successful **then**
9: Now C is said to have established a verified connection with C'.
10: **else**
11: C dissolves; its members request reconnection.
12: **end if**

Algorithm 11 Verified Random Bit Fixing. This procedure is initiated by each committee C at every round and takes $O(\texttt{dim}\log n)$ rounds to complete. A single invocation of this procedure yields $\Theta(\log n)$ random committee connections for C. When invoked in a pipelined manner every round, C gets a steady stream of $\Theta(\log n)$ random committee connections every round as long as C belongs to a healthy committee.

1: Generate $\Theta(\log n)$ random $\texttt{dim}$-bit binary strings.
2: **for** Each $s \in S$ in parallel **do**
3: Let $C_{\texttt{dim}}$ be the committee at hypercube location s.
4: $(C = C_0, C_1, \ldots, C_i, \ldots, C_{\texttt{dim}-1}) \triangleq$ sequence of committees in the bit-fixing path from C to $C_{\texttt{dim}}$.
5: **for** $i = 0$ to $\texttt{dim} - 1$ **do**
6: C requests verified connection to C_{i+1} with C_i's introduction (see Algorithm 10).
7: **end for**
8: **end for**
9: The set of final committees reached in parallel is the required set of verified random committee connections.

Algorithm 12 Shuffle. This procedure is continually executed in the background.

1: Whenever a node u becomes a member of a committee C, C starts a timer for some polylog(n) rounds.

2: Once the timer expires, C initiates Byzantine agreement to get the committee to agree on a round r when it should perform handover.

3: At round r, C initiates the handover of u to a new random committee C' that C received in the most recent round via **Verified Random Bit Fixing**. See Algorithm 13.

4: Note that u may need to maintain temporary membership in C in order to complete any other Byzantine Agreement that started when u still belonged to C. So u will need to maintain dual membership during its transition.

Algorithm 13 Handover. This procedure is initiated individually by members of a committee C that wishes to pass on a node u to another committee C'. Appropriate preconditions must, however, be met. This can be invoked only if C' was a verified random committee obtained by C in the most recent round. Furthermore, members of C must have agreed to handover u in the current round. Alternatively, one of the members must claim u to be a new node that was attached to it. Note that a node can at most claim $O(1)$ such new nodes each round.

1: Each member of C individually checks if the preconditions are appropriately met; if not, it does not proceed.

2: Each member of C sends a message to every node in C' indicating u's transfer to C'.

3: Each member of C' immediately accepts u if at least $(9/10)$-fraction of the nodes in C sent a consistent message. This handover is effective immediately even without Byzantine agreement, which is not required because C has already established a verified connection with C'.

Algorithm 14 Integrate. This procedure is invoked whenever the adversary connects a node u to a node v in some committee C.

Node v informs every member of C that u is a new node.

The committee waits for a sufficiently long (but $O(\text{polylog}(n))$ rounds (long enough for a new invocation of Algorithm 11 to have started and completed).

Members of C invoke the handover protocol (see Algorithm 13).

Appendices

4.A The Byzantine Sampling Theorem in Regular, Sparse Networks with Constant Expansion

We describe and analyze a mechanism to achieve an *almost uniform, random sampling* of nodes in a static network with Byzantine nodes. We closely emulate the works of [7] and [8] and modify them suitably to serve our purpose. See Algorithm 4 for the exact mechanism of the random sampling procedure.

Remark 15. Our algorithm requires nodes to initiate $h\log^p n$ random walks simultaneously, for fixed (constant) positive integers h and p. Here is where we differ slighly from [7] — their work dictated that nodes initiate $h\log n$ random walks simultaneously.

Remark 16. The length of the random walks τ is defined as $\tau \stackrel{\text{def}}{=} M\log n$, where M is a large enough positive constant.

During the run of the algorithm, a node u might receive a large number of tokens (possibly generated by Byzantine nodes). Nevertheless, node u only forwards at most $h\log^p n$ tokens in each step in order to ensure (whp) passage of random walks that matter to us. More precisely, the random walks that arrive at a node u are placed in a FIFO

buffer according to the order of their arrival. In every round, node u sends out at most $h\log^p n$ top elements in the buffer.

Definition 18. *We say that a node v is "affected" if v is honest and it is a neighbor of some Byzantine node. An honest node that is not "affected" is called "unaffected".*

Each Byzantine node is connected to at most d honest nodes. Thus the number of affected nodes is at most $d\beta(n) = \frac{dn}{\log^\ell n}$. Thus if we denote the set of unaffected nodes by `Unaffected`, then

Observation 17. $|\texttt{Unaffected}| \geq n - \frac{(d+1)n}{\log^\ell n}$.

Definition 19. *If a random walk only ever visits unaffected nodes up to (but excluding) round t, we say that the walk is "good" until round t and becomes bad from round $t+1$ on. A walk that never visits any affected node or any Byzantine node is simply called "good".*

Before delving into the proof of Theorem 7 we introduce some technical machinery. Our proof will use the relationship between the spectral gap and vertex expansion of a graph.

Consider a graph G of n nodes and a set $S \subset V(G)$. Let $N_G(S)$ be the set of neighbors of the nodes in S in $G \setminus S$. The vertex expansion ϕ of G is defined as

$$\phi \stackrel{\text{def}}{=} \min\left\{ \frac{|N_G(S)|}{|S|} \ : \ S \subset V(G) \text{ and } |S| \leq \frac{n}{2} \right\}.$$

Theorem 11 (Cheeger Inequality, [57])**.** *Let G be a d-regular graph with spectral gap $1-\lambda$ and vertex expansion ϕ. Then*

$$\frac{1-\lambda}{2} \leq \phi \leq \sqrt{2(1-\lambda)}.$$

In the proof of Theorem 7 we would like to argue that, after removing nodes *affected* by Byzantine nodes from our expander G, the remaining network still contains a subgraph H that has constant expansion and is of size $nO(\beta(n)) = n - O(\frac{n}{\log^{\ell} n})$.

This motivates us to extend the interesting result of Theorem 2.1 in [17], which states that, upon removing a set F of size εn nodes from an n-node network G with constant expansion and constant node degree, there is a subgraph H of size $n - \varepsilon' n$ contained in the remainder graph $G_F \subset G \setminus F$ that itself has constant expansion; ε and ε' are small positive constants. Note that we cannot use the result of [17] for our purposes directly, as this would yield a core set of size $n - \Theta(n)$ whereas we require the expander component to be of size $\geq n - O(\frac{n}{\log^{\ell} n})$. The analysis of this result uses a pruning procedure `Prune(c)` that iteratively removes sets of nodes that have small expansion from the residual graph G_F, yielding a sequence of pruned graphs $G_F = G_0 \supset G_1 \supset \cdots \supset G_m = H$. Thus we have the following result.

Lemma 48 (Lemma 3 in [9])**.** *Let G be an n-node graph with vertex expansion ϕ and constant node degree d and suppose that all nodes in a set F are removed from G, where $F \subset G$ and $|F| = o(n)$. Then, for any positive constant $c < 1$, there exists a subgraph H of G such that*

1. *H has vertex expansion $c\phi$, and*
2. *$|H| \geq n - |F|(1 + \frac{1}{\phi(1-c)})$.*

The following lemma bounds the time that it takes for most random walks to complete their $\Theta(\log n)$ steps (required for mixing). Its proof follows along the same lines as Lemma 2 in [7].

Lemma 49. *Consider a static, d-regular, non-bipartite, expander network $G = (V,E)$ with at most $B(n) = \frac{n}{\log^{\ell} n}$ Byzantine nodes. If each honest node $v \in V$ starts $h\log^p n$ random walks of length $\tau = M\log n$, then, with high probability, every token ρ that originated from a node $u \in$ `Unaffected` and only reached unaffected nodes, can complete τ steps in $\tau(\tau+1)$ rounds. Moreover, ρ remains no longer than 2τ rounds at the same node.*

Here p is a fixed positive constant and M is a sufficiently large, fixed positive constant.

Proof. Since G is d-regular, the expected number of tokens received by any unaffected node in any round is $\leq h\log^p n$. By applying a standard Chernoff bound, it follows that with probability $\geq 1-n^{-7}$, each node in `Unaffected` receives at most $2h\log^p n$ tokens in any particular round. Taking a union bound over $2\tau^2 = 2M^2\log^2 n = O(\log^2 n)$ rounds and all unaffected nodes, the same is true at every node in `Unaffected` during rounds $[1,2\tau^2]$ with probability $\geq 1-n^{-5}$.

According to our token forwarding algorithm, tokens are forwarded using a FIFO policy. Moreover, each node forwards up to $h\log^p n$ tokens per round and thus it follows that any token arriving at a node u in round i, can be delayed for at most $2i$ additional rounds. It follows that, after

$$\sum_{i=1}^{\tau} 2i = \tau(\tau+1) < 2\tau^2 \in O(\log^2 n)$$

rounds, all good tokens will have taken τ steps with high probability. □

Remark 17. While the above lemma is in essence quite similar to Lemma 2 in [7], it is different in one important aspect: We generalize the number of random walks to $\Theta(\log^p n)$ from $\Theta(\log n)$ for any arbitrary, fixed positive constant p.

We are now ready to prove the Byzantine sampling theorem:

Theorem 12 (Byzantine Random Sampling)**.** *Let $T_{walk} = \tau(\tau+1) = \Theta(\log^2 n)$ and consider an n-node (static) expander network G under an oblivious (Byzantine) adversary, and suppose that at most $\beta(n)$ nodes are Byzantine, where $\beta(n) \stackrel{\text{def}}{=} \frac{n}{\log^\ell n}$ nodes are Byzantine, where ℓ is a sufficiently large constant. Also suppose that each of the honest nodes introduces $h\log^p n$ random walk tokens at time zero (for some suitable positive constants h and p) and those random walks are executed according to the mechanism described previously.*

Then there exists a set of honest nodes `Core` *of size $\geq n - O(\frac{n}{\log^\ell n})$ such that, in every time interval $[iT_{walk}+1, (i+1)T_{walk}]$ for $0 \leq i \leq \Theta(\log n)$, the following hold:*

1. *A random walk token originating from a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to terminate at any particular node in* `Core`*.*

2. *A random walk token that terminates at a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to have originated at any particular node in* `Core`*.*

3. *At most $O(\frac{n}{\log^{\ell-p-2} n})$ nodes in* `Core` *receive tokens that did not originate in* `Core` *or did not take all their steps among nodes in* `Core`*.*

Proof. As a first step, we describe the set `Core` and bound its size: Let B' denote the set of Byzantine nodes together with the set of *affected* nodes, i.e., let $B' \stackrel{\text{def}}{=} B \cup \texttt{Affected}$. It follows from Observation 17 that $|B'| \leq \frac{(d+1)n}{\log^k n}$.

We know from Lemma 48 that, upon removing set B' from an n-node network with constant vertex expansion and constant node degree, there is a subgraph H of size $n - O(|B'|)$ in the remainder graph (the subgraph of G induced by $V \setminus B'$) such that H itself

has constant vertex expansion. Note that, by Cheeger's inequality (see Theorem 11), we know that H also has constant spectral gap. We define $\texttt{Core} \stackrel{\text{def}}{=} H$ and observe that $|\texttt{Core}| \geq n - O(|B'|) = n - O(|B|) = n - O(\beta(n))$.

Next, we bound the number of tokens that are received by nodes in Core in any particular interval $[iT_{walk}+1, (i+1)T_{walk}]$: Recall that nodes in Core are never adjacent to Byzantine nodes. Thus, any token that is sent across a link of the cut $(\texttt{Core}, V \setminus \texttt{Core})$ was sent by an honest node. It follows from Lemma 49 that (whp) at most $2h\log^p n \cdot T_{walk} = 2h\log^p n \cdot \tau(\tau+1) = 2h\log^p n \cdot M\log n(M\log n+1) = O(\log^{p+2} n)$ tokens are sent across any edge during $[iT_{walk}, (i+1)T_{walk}]$. The size of the cut $(\texttt{Core}, V \setminus \texttt{Core})$ is bounded by $d(n|Core|)$ and thus at most $O(\log^{p+2} n) \cdot d(n|Core|) = O(\beta(n) \cdot \log^{p+2} n) = O(\frac{n}{\log^{\ell-p-2} n})$ tokens originating from outside Core reach nodes in Core and vice versa. Thus at most $O(\frac{n}{\log^{\ell-p-2} n})$ nodes in Core receive tokens that did not originate in Core or did not take all their steps in Core. This proves Clause 3 of the theorem statement.

Finally, we show that the tokens that remained in Core are almost uniformly distributed on Core after completing their τ steps: For the sake of our analysis, we consider a (virtual) graph $\bar{G}$ that consists of the same honest nodes as G, with the differences that

1. we replace the Byzantine nodes by honest nodes, and
2. in contrast to the actual network G, we assume that edges in the virtual network $\bar{G}$ have infinite bandwidth.

We use the notation $\bar{v_j}$ to refer to the node in $\bar{G}$ that corresponds to $v_j \in G$. Recall that $\bar{G}$ is a d-regular expander graph. Thus it follows from [85] that, after taking $\tau = M\log n$ steps, for a sufficiently large constant M, a random walk token is at any particular node in

$\bar{G}$ with probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$. Note that the above remains true even if we delay the token at every visiting node for some number of rounds, as long as the token takes τ steps in total.

Consider a token ρ generated in round $iT_{walk} + 1$ and suppose that ρ terminated at some node v_τ by round $(i+1)T_{walk}$. We define $(v_0, v_1, v_2, \ldots, v_{\tau-1}, v_\tau)$ to be the sequence of tokens visited by ρ, where each $v_j \in \texttt{Core}$ for $0 \leq j \leq \tau$. Let $(r_0, r_1, r_2, \ldots, r_{\tau-1}, r_\tau)$ denote the sequence of rounds such that ρ is forwarded from v_j to v_{j+1} in round r_j, for $0 \leq j \leq \tau$. In the remainder of the proof, we show that v_τ is almost uniformly distributed among the nodes in $\texttt{Core}$.

Consider a token $\bar{\rho}$ that terminated at node $\bar{v_\tau}$ in the virtual graph $\bar{G}$, and suppose that the random sources of $\bar{v_j} \in \bar{G}$ and $v_j \in G$ are coupled, for $0 \leq j \leq \tau$. In round iT_{walk}, node $\bar{v_0}$ holds $\bar{\rho}$ and, due to the coupling of the random sources, $\bar{v_0}$ forwards $\bar{\rho}$ to the neighbor $\bar{v_1}$ (corresponding to v_1) in round r_0. Node $\bar{v_1}$ in turn delays $\bar{\rho}$ until round r_1 and then forwards $\bar{\rho}$ to neighbor $\bar{v_2}$. And so forth.

Hence, $\bar{\rho}$ takes the exact same sequence of steps in $\bar{G}$ as it does in G. It follows that $\bar{\rho}$ is almost uniformly distributed among $\bar{\texttt{Core}}$, and thus the probability of ρ being at any particular node in $\texttt{Core}$ is in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$.

This proves Clause 1 of the theorem statement.

Since G is a regular graph, we can use the reversibility of random walks and use similar arguments as above to show Clause 2 of the theorem statement. □

This yields Theorem 7 as an immediate corollary.

Corollary 2 (Theorem 7)**.** *Let $T_{walk} = \tau(\tau+1) = \Theta(\log^2 n)$ and consider an n-node*

(static) expander network G under an oblivious (Byzantine) adversary, and suppose that at most $\beta(n)$ nodes are Byzantine, where $\beta(n) \stackrel{\text{def}}{=} \frac{n}{\log^{\ell} n}$. Also suppose that each of the honest nodes introduces $c_2 \log^2 n$ random walk tokens at time zero (for some suitable positive constant c_2) and those random walks are executed according to the mechanism described previously.

Then there exists a set of honest nodes `Core` *of size $\geq n - O(\frac{n}{\log^{\ell} n})$ such that, in every time interval $[iT_{walk} + 1, (i+1)T_{walk}]$ for $0 \leq i \leq \Theta(\log n)$, the following hold:*

1. *A random walk token originating from a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to terminate at any particular node in* `Core`*.*

2. *A random walk token that terminates at a node in* `Core` *has probability in $[\frac{1}{n} - \frac{1}{n^5}, \frac{1}{n} + \frac{1}{n^5}]$ to have originated at any particular node in* `Core`*.*

3. *At most $O(\frac{n}{\log^{\ell-4} n})$ nodes in* `Core` *receive tokens that did not originate in* `Core` *or did not take all their steps among nodes in* `Core`*.*

Chapter 5

Conclusion and Future Directions

In this dissertation, we looked at three fundamental problems in distributed computing, namely, *leader election*, *Byzantine counting*, and *Byzantine routing*. In each of these problems, we focused on a specific performance metric and designed efficient algorithms vis-a-vis that metric.

First we looked at the leader election problem in diameter-two networks, for which we developed time- and message-optimal algorithms — both randomized and deterministic. We also proved matching lower bounds showing that our algorithms are the best possible (asymptotically speaking) in terms of time and message complexity.

The second problem that we considered was that of network size estimation in the presence of Byzantine faults. We designed a fast[1] and lightweight algorithm that achieves "almost everywhere", approximate knowledge of the network size in the presence of a large number of Byzantine faults. In doing so, we analyzed an important model of random graphs — namely the $H(n,d)$-model of random graphs (see Section 3.A) — and derived

[1] running in polylog(n) rounds

some fascinating properties. In particular, we showed that such a random graph "looks like" a tree (see Lemma 29 in Chapter 3) locally. We used these properties crucially in proving the correctness of our algorithm. We believe that our work paves the way for new works in this area that could build not only on our result, but also on our techniques. The Byzantine counting algorithm is a crucial subroutine that is needed as a primitive in many distributed computing tasks, and our analysis of the aforementioned random graph model can be used in many other problems as well.

The third and the final problem we looked at was that of building a routing structure that is provably *robust* to the presence of a large number of Byzantine (malicious) nodes as well as to large dynamism in the network. We show how to build a hypercube — a network structure with three desired characteristics built in it, namely

1. sparsity (each node has only polylog (n) neighbors),
2. good vertex expansion, and
3. routability (by means of bit-fixing).

This hypercube enables the construction of a *distributed hash table* or a *DHT* and can also serve as a network backbone for solving many other fundamental distributed computing problems such as *Byzantine agreement* and *Byzantine leader election*.

This dissertation highlights the fact that random walk techniques are very useful in static networks as well as in dynamic networks. We believe that the research presented in this dissertation opens up a lot of interesting directions for further study. We have discussed this at the end of each chapter. We would like to emphasize here on some of them.

The very first problem considered, namely leader election in diameter-two networks, naturally begs the question whether other fundamental distributed computing tasks such as performing all-to-all *routing* (or even *permutation routing*), building a *minimum spanning tree* (*MST*), etc. can be done in a time- and message-optimal way in diameter-two networks. These problems have been extensively studied in the context of diameter-one networks (i.e., complete graphs) but remain largely unexplored in the case of diameter-two networks.

The problem of Byzantine counting can be extended to more general (i.e., less restrictive) network models. One can also try to design *deterministic* algorithms to achieve "almost everywhere", approximate counting. This seems to be a promising direction; my coauthors and I have done this work [25] and have submitted it for review. One can also focus on increasing the *tolerance* of the algorithm vis-a-vis Byzantine nodes, e.g., it would be great to have the algorithm tolerate up to $\frac{n}{\text{polylog}(n)}$ Byzantine nodes, compared to the current tolerance of $n^{1-\delta}$ (where δ is any arbitrarily small, but fixed, positive constant) Byzantine nodes.

In the case of building routable structures robust to Byzantine nodes and heavy network churn, one may consider constructing other, non-hypercube structures with higher vertex expansion and lower average node degrees. For example, one might aim to achieve a constant average node degree along with a constant vertex expansion.

References

[1] Yehuda Afek, Baruch Awerbuch, and Eli M. Gafni. "Applying Static Network Protocols to Dynamic Networks". In: *28th Annual Symposium on Foundations of Computer Science*. FOCS '87. 1987, pp. 358–370.

[2] Yehuda Afek and Eli Gafni. "Time and Message Bounds for Election in Synchronous and Asynchronous Complete Networks". In: *SIAM Journal of Computing* 20.2 (1991), pp. 376–394.

[3] Yehuda Afek, Eli M. Gafni, and Adi Rosén. "The Slide Mechanism with Applications in Dynamic Networks". In: *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*. PODC '92. Vancouver, British Columbia, Canada: ACM, 1992, pp. 35–46.

[4] Lorenzo Alvisi, Dahlia Malkhi, Evelyn Pierce, and Michael K. Reiter. "Fault Detection for Byzantine Quorum Systems". In: *IEEE Transactions on Parallel and Distributed Systems* 12.9 (2001), pp. 996–1007.

[5] John Augustine, Anisur Rahaman Molla, Ehab Morsy, Gopal Pandurangan, Peter Robinson, and Eli Upfal. "Storage and Search in Dynamic Peer-to-peer Networks". In: *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '13. Montréal, Québec, Canada: ACM, 2013, pp. 53–62.

[6] John Augustine, Gopal Pandurangan, and Peter Robinson. "Distributed Algorithmic Foundations of Dynamic Networks". In: *SIGACT News* 47.1 (2016), pp. 69–98.

[7] John Augustine, Gopal Pandurangan, and Peter Robinson. "Fast Byzantine Agreement in Dynamic Networks". In: *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*. PODC '13. Montréal, Québec, Canada: ACM, 2013, pp. 74–83.

[8] 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*. Ed. by Yoram Moses. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 276–291.

[9] 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*. 2015, pp. 350–369.

[10] 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 '12. Kyoto, Japan: Society for Industrial and Applied Mathematics, 2012, pp. 551–569.

[11] 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*. 2018, pp. 1060–1069.

[12] Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Michael Saks. "Adapting to Asynchronous Dynamic Networks (Extended Abstract)". In: *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*. STOC '92. Victoria, British Columbia, Canada: ACM, 1992, pp. 557–570.

[13] 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 '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, pp. 318–327.

[14] Baruch Awerbuch and Christian Scheideler. "Towards a Scalable and Robust DHT". In: *Theory of Computing Systems* 45.2 (2009), pp. 234–260.

[15] Baruch Awerbuch and Michael Sipser. "Dynamic Networks Are As Fast As Static Networks". In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. FOCS '88. 1988, pp. 206–219.

[16] Ozalp Babaoglu, Moreno Marzolla, and Michele Tamburini. “Design and Implementation of a P2P Cloud System”. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC ’12. Trento, Italy: ACM, 2012, pp. 412–417.

[17] Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. “The Effect of Faults on Network Expansion”. In: *Theory of Computing Systems* 39.6 (2006), pp. 903–928.

[18] Marc Barthélémy and Luıs A. Nunes Amaral. “Small-world Networks: Evidence for a Crossover Picture”. In: *Physical Review Letters* 82 (15 1999), pp. 3180–3183.

[19] *Bittorrent Sync*. `https://www.resilio.com/`. Accessed: 2018-07-15.

[20] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. “BRAHMS: Byzantine Resilient Random Membership Sampling”. In: *Computer Networks* 53.13 (2009). Preliminary version in PODC 2008., pp. 2340–2359.

[21] Ruud van de Bovenkamp, Fernando A. Kuipers, and Piet Van Mieghem. “Gossip-based Counting in Dynamic Networks”. In: *NETWORKING 2012 - 11th International IFIP TC 6 Networking Conference, Prague, Czech Republic, May 21-25, 2012, Proceedings, Part II*. 2012, pp. 404–417.

[22] John Canny. “Collaborative Filtering with Privacy”. In: *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. SP ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 45–57.

[23] Anthony Chadd. “DDoS Attacks: Past, Present, and Future”. In: *Network Security* 2018.7 (2018), pp. 13–15.

[24] Yu-Wei Chan, Tsung-Hsuan Ho, Po-Chi Shih, and Yeh-Ching Chung. “Malugo: A Peer-to-peer Storage System”. In: *International Journal of Ad Hoc and Ubiquitous Computing* 5.4 (2010), pp. 209–218.

[25] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. “Keeping Score Against Evildoers: Byzantine-Resilient Counting in Networks”. *(under submission)*. 2019.

[26] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. “Network Size Estimation in Small-world Networks under Byzantine Faults”. In: *Proceedings*

of the 34th IEEE International Parallel and Distributed Processing Symposium. IPDPS '19. 2019.

[27] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. "The Complexity of Leader Election in Diameter-two Networks". In: *Distributed Computing* (2019). *(in press)*.

[28] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. "The Complexity of Leader Election: A Chasm at Diameter Two". In: *Proceedings of the 19th International Conference on Distributed Computing and Networking*. ICDCN '18. Varanasi, India: ACM, 2018, 13:1–13:10.

[29] Lin Chen, Amin Karbasi, and Forrest W. Crawford. "Estimating the Size of a Large Network and Its Communities from a Random Sample". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 3072–3080.

[30] Bogdan S. Chlebus and Dariusz R. Kowalski. "Locally Scalable Randomized Consensus for Synchronous Crash Failures". In: *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*. SPAA '09. Calgary, AB, Canada: ACM, 2009, pp. 290–299.

[31] Nicolas Christin, Andreas S. Weigend, and John Chuang. "Content Availability, Pollution and Poisoning in File-sharing Peer-to-peer Networks". In: *Proceedings of the Sixth ACM Conference on Electronic Commerce*. EC '05. Vancouver, BC, Canada: ACM, 2005, pp. 68–77.

[32] *Cloudmark*. `https://www.cloudmark.com/en`. Accessed: 2018-07-15.

[33] Souptik Datta, Kanishka Bhaduri, Chris Giannella, Ran Wolff, and Hillol Kargupta. "Distributed Data Mining in Peer-to-peer Networks". In: *IEEE Internet Computing* 10.4 (2006), pp. 18–26.

[34] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and H. Raymond Strong. "An Efficient Algorithm for Byzantine Agreement Without Authentication". In: *Information and Control* 52.3 (1982), pp. 257–274.

[35] Shlomi Dolev. *Self-stabilization*. Cambridge, MA, USA: MIT Press, 2000.

[36] Peter Druschel and Antony Rowstron. “PAST: A Large-scale, Persistent Peer-to-peer Storage Utility”. In: *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*. 2001, pp. 75–80.

[37] Peter Druschel and Antony Rowstron. “Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility”. In: *ACM SIGOPS Operating Systems Review* 35.5 (2001), pp. 188–201.

[38] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. 1st. New York, NY, USA: Cambridge University Press, 2009.

[39] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. “Fault Tolerance in Networks of Bounded Degree”. In: *SIAM Journal on Computing* 17.5 (1988), pp. 975–988.

[40] David A. Easley and Jon M. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[41] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. “Profiling a Million User DHT”. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC ’07. San Diego, California, USA: ACM, 2007, pp. 129–134.

[42] Amos Fiat and Jared Saia. “Censorship Resistant Peer-to-peer Content Addressable Networks”. In: *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’02. San Francisco, California: Society for Industrial and Applied Mathematics, 2002, pp. 94–103.

[43] Amos Fiat and Jared Saia. “Censorship Resistant Peer-to-peer Networks”. In: *Theory of Computing* 3.1 (2007), pp. 1–23.

[44] *Filecoin*. `https://filecoin.io/`. Accessed: 2018-10-27.

[45] *Freenet*. `https://freenetproject.org/`. Accessed: 2018-07-16.

[46] Joel Friedman. “On the Second Eigenvalue and Random Walks in Random d-regular Graphs”. In: *Combinatorica* 11.4 (1991), pp. 331–362.

[47] Eli M. Gafni and Dimitri P. Bertsekas. “Distributed Algorithms for Generating Loop-free Routes in Networks with Frequently Changing Topology”. In: *IEEE Transactions on Communications* 29.1 (1981), pp. 11–18.

[48] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, Erwan Le Merrer, and Laurent Massoulié. "Peer Counting and Sampling in Overlay Networks Based on Random Walks". In: *Distributed Computing* 20.4 (2007), pp. 267–278.

[49] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. "Vanish: Increasing Data Privacy with Self-destructing Data". In: *Proceedings of the 18th Conference on USENIX Security Symposium*. SSYM'09. Montréal, Canada: USENIX Association, 2009, pp. 299–316.

[50] Seth Gilbert, Peter Robinson, and Suman Sourav. "Slow Links, Fast Links, and the Cost of Gossip". In: *2018 IEEE 38th International Conference on Distributed Computing Systems*. ICDCS '18. 2018, pp. 786–796.

[51] Fabiola Greve, Murilo Santos de Lima, Luciana Arantes, and Pierre Sens. "A Time-Free Byzantine Failure Detector for Dynamic Networks". In: *2012 Ninth European Dependable Computing Conference*. EDCC 2012. 2012, pp. 191–202.

[52] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. "Highly Dynamic Distributed Computing with Byzantine Failures". In: *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*. PODC '13. Montréal, Québec, Canada: ACM, 2013, pp. 176–183.

[53] 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". In: *SIGCOMM Computer Communication Review* 32.1 (2002), pp. 82–82.

[54] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. "The Case for Byzantine Fault Detection". In: *Proceedings of the Second Conference on Hot Topics in System Dependability – Volume 2*. HOTDEP 2006. Seattle, WA: USENIX Association, 2006, pp. 5–5.

[55] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy Campbell. "A Survey of Peer-to-peer Storage Techniques for Distributed File Systems". In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*. ITCC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 205–213.

[56] Kirsten Hildrum and John Kubiatowicz. "Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-peer Networks". In: *Distributed Computing: 17th International Conference, DISC 2003, Sorrento, Italy, October 1-3, 2003. Proceed-*

ings. Ed. by Faith Ellen Fich. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 321–336.

[57] Shlomo Hoory, Nathan Linial, and Avi Wigderson. "Expander Graphs and Their Applications". In: *Bulletin of the American Mathematical Society* 43.4 (2006), pp. 439–561.

[58] Keren Horowitz and Dahlia Malkhi. "Estimating Network Size from Local Information". In: *Information Processing Letters* 88.5 (2003), pp. 237–243.

[59] Pierre A. Humblet. "Electing a Leader in a Clique in $O(n \log n)$ Messages". Intern. Memo., Laboratory for Information and Decision Systems, MIT, Cambridge, Massachusetts. 1984.

[60] Riko Jacob, Andrea Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. "SKIP+: A Self-stabilizing Skip Graph". In: *Journal of the ACM* 61.6 (2014), 36:1–36:26.

[61] Tim Jacobs and Gopal Pandurangan. "Stochastic Analysis of a Churn-tolerant Structured Peer-to-peer Scheme". In: *Peer-to-peer Networking and Applications* 6.1 (2013), pp. 1–14.

[62] Sidharth Jaggi, Michael Langberg, Sachin Katti, Tracey Ho, Dina Katabi, Muriel Médard, and Michelle Effros. "Resilient Network Coding in the Presence of Byzantine Adversaries". In: *IEEE Transactions on Information Theory* 54.6 (2008), pp. 2596–2603.

[63] M. Frans Kaashoek and David R. Karger. "Koorde: A Simple Degree-optimal Distributed Hash Table". In: *Peer-to-peer Systems II*. Ed. by M. Frans Kaashoek and Ion Stoica. IPTPS '03. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 98–107.

[64] Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. "Fast Asynchronous Byzantine Agreement and Leader Election with Full Information". In: *ACM Transactions on Algorithms* 6.4 (2010), 68:1–68:28.

[65] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. "Efficient Distributed Approximation Algorithms via Probabilistic Tree Embeddings". In: *Distributed Computing* 25.3 (2012), pp. 189–205.

[66] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. "Byzantine Fault Detectors for Solving Consensus". In: *The Computer Journal* 46.1 (2003), pp. 16–35.

[67] Valerie King and Jared Saia. "Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary". In: *Journal of the ACM* 58.4 (2011), 18:1–18:24.

[68] Valerie King and Jared Saia. "Faster Agreement via a Spectral Method for Detecting Malicious Behavior". In: *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '14. Portland, Oregon: Society for Industrial and Applied Mathematics, 2014, pp. 785–800.

[69] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. "Scalable Leader Election". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. SODA '06. Miami, Florida: Society for Industrial and Applied Mathematics, 2006, pp. 990–999.

[70] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. "Scalable Leader Election". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '06. Miami, Florida: Society for Industrial and Applied Mathematics, 2006, pp. 990–999.

[71] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. "Towards Secure and Scalable Computation in Peer-to-peer Networks". In: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 87–98.

[72] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. "Towards Secure and Scalable Computation in Peer-to-peer Networks". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 87–98.

[73] Ephraim Korach, Shay Kutten, and Shlomo Moran. "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12.1 (1990), pp. 84–101.

[74] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. "Optimal Lower Bounds for Some Distributed Algorithms for a Complete Network of Processors". In: *Theoretical Computer Science* 64.1 (1989), pp. 125–132.

[75] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. “The Optimality of Distributive Constructions of Minimum Weight and Degree Restricted Spanning Trees in a Complete Network of Processors”. In: *SIAM Journal on Computing* 16.2 (1987), pp. 231–236.

[76] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. “Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors”. In: *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*. PODC ’84. Vancouver, British Columbia, Canada: ACM, 1984, pp. 199–207.

[77] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. “Distributed Computation in Dynamic Networks”. In: *Proceedings of the Forty-second ACM Symposium on Theory of Computing*. STOC ’10. Cambridge, Massachusetts, USA: ACM, 2010, pp. 513–522.

[78] Fabian Kuhn and Rotem Oshman. “Dynamic Networks: Models and Algorithms”. In: *ACM SIGACT News* 42.1 (2011), pp. 82–96.

[79] Fabian Kuhn, Stefan Schmid, and Roger Wattenhofer. “Towards Worst-case Churn-resistant Peer-to-peer Systems”. In: *Distributed Computing* 22.4 (2010), pp. 249–267.

[80] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. “On the Complexity of Universal Leader Election”. In: *Journal of the ACM* 62.1 (2015). Invited paper from *ACM PODC* 2013, 7:1–7:27.

[81] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. “Sublinear Bounds for Randomized Leader Election”. In: *Theoretical Computer Science* 561, Part B (2015). Special Issue on Distributed Computing and Networking, pp. 134–143.

[82] Ching Law and Kai-Yeung Siu. “Distributed Construction of Random Expander Networks”. In: *Proceedings IEEE INFOCOM 2003, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, San Franciso, CA, USA, March 30 - April 3, 2003*. 2003, pp. 2133–2143.

[83] Gérard Le Lann. “Distributed Systems — Towards a Formal Approach”. In: *IFIP Congress*. 1977, pp. 155–160.

[84] James Li. *A Survey of Peer-to-peer Network Security Issues*. `https://www.cse.wustl.edu/~jain/cse571-07/ftp/p2p/`. Accessed: 2018-10-10. 2007.

[85] László Lovász. "Random Walks on Graphs: A Survey". In: *Combinatorics, Paul Erdős is eighty* 2 (1993), pp. 1–46.

[86] Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. "Counting in Anonymous Dynamic Networks Under Worst-case Adversary". In: *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*. ICDCS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 338–347.

[87] Nancy A. Lynch. *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[88] Peter Mahlmann and Christian Schindelhauer. "Peer-to-peer Networks Based on Random Transformations of Connected Regular Undirected Graphs". In: *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '05. Las Vegas, Nevada, USA: ACM, 2005, pp. 155–164.

[89] 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 '05. Fairfax, VA, USA: ACM, 2005, pp. 72–80.

[90] Dahlia Malkhi, Moni Naor, and David Ratajczak. "Viceroy: A Scalable and Dynamic Emulation of the Butterfly". In: *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*. PODC '02. Monterey, California: ACM, 2002, pp. 183–192.

[91] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric". In: *Peer-to-peer Systems*. Ed. by Peter Druschel, Frans Kaashoek, and Antony Rowstron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65.

[92] Silvio Micali and Tal Rabin. "Collective Coin Tossing Without Assumptions nor Broadcasting". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. 1990, pp. 253–266.

[93] Rich Miller. *P2P Networks Hijacked for DDoS Attacks*. https://news.netcraft.com/archives/2007/05/23/p2p_networks_hijacked_for_ddos_attacks.html. Accessed: 2018-10-10. 2007.

[94] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. 2nd edition. Cambridge CB2 8BS, United Kingdom: Cambridge University Press, 2017.

[95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. New York, NY, USA: Cambridge University Press, 1995.

[96] Moni Naor and Udi Wieder. "A Simple Fault-tolerant Distributed Hash Table". In: *Peer-to-peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers*. Ed. by M. Frans Kaashoek and Ion Stoica. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 88–97.

[97] Moni Naor and Udi Wieder. "Scalable and Dynamic Quorum Systems". In: *Distributed Computing* 17.4 (2005), pp. 311–322.

[98] Mikhail Nesterenko and Sébastien Tixeuil. "Discovering Network Topology in the Presence of Byzantine Faults". In: *Structural Information and Communication Complexity: 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006. Proceedings*. Ed. by Paola Flocchini and Leszek Gąsieniec. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 212–226.

[99] Calvin Newport and Peter Robinson. "Fault-Tolerant Consensus with an Abstract MAC Layer". In: *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*. 2018, 38:1–38:20.

[100] Shreyas Pai, Gopal Pandurangan, Sriram V. Pemmaraju, Talal Riaz, and Peter Robinson. "Symmetry Breaking in the Congest Model: Time- and Message-efficient Algorithms for Ruling Sets". In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. 2017, 38:1–38:16.

[101] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. "Building Low-diameter Peer-to-peer Networks". In: *IEEE Journal on Selected Areas in Communications* 21.6 (2003), pp. 995–1002.

[102] Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. "DEX: self-healing Expanders". In: *Distributed Computing* 29.3 (2016), pp. 163–185.

[103] Gopal Pandurangan and Amitabh Trehan. "Xheal: A Localized Self-healing Algorithm Using Expanders". In: *Distributed Computing* 27.1 (2014), pp. 39–54.

[104] Arjan J. H. Peddemors. *Cloud Storage and Peer-to-peer Storage: End-user Considerations and Product Overview*. Accessed: 2018-07-15. 2010.

[105] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9780898719772`.

[106] David Peleg. "Time-optimal Leader Election in General Networks". In: *Journal of Parallel and Distributed Computing* 8.1 (1990), pp. 96–99.

[107] Radia Perlman. *Routing with Byzantine Robustness*. Tech. rep. Sun Microsystems, Inc., Mountain View, CA, USA, 2005.

[108] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. "A Scalable Content-addressable Network". In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 161–172.

[109] Antony I. T. Rowstron and Peter Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-peer Systems". In: *Proceedings of the IFIP-ACM International Conference on Distributed Systems Platforms Heidelberg*. Middleware '01. London, UK: Springer-Verlag, 2001, pp. 329–350.

[110] Nicola Santoro. *Design and Analysis of Distributed Algorithms (Wiley Series on Parallel and Distributed Computing)*. New York, NY, USA: Wiley-Interscience, 2006.

[111] Christian Scheideler. "How to Spread Adversarial Nodes?: Rotate!" In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA: ACM, 2005, pp. 704–713.

[112] Christian Scheideler and Stefan Schmid. "A Distributed and Oblivious Heap". In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*. ICALP '09. Rhodes, Greece: Springer-Verlag, 2009, pp. 571–582.

[113] Subhabrata Sen and Jia Wang. "Analyzing Peer-to-peer Traffic Across Large Networks". In: *IEEE/ACM Transactions on Networking (TON)* 12.2 (2004), pp. 219–232.

[114] Tallat M. Shafaat, Ali Ghodsi, and Seif Haridi. "A Practical Approach to Network Size Estimation for Structured Overlays". In: *Self-Organizing Systems: Third International Workshop, IWSOS 2008, Vienna, Austria, December 10-12, 2008. Proceedings*. Ed. by Karin Anna Hummel and James P. G. Sterbenz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 71–83.

[115] *Sia*. `https://sia.tech/`. Accessed: 2018-10-27.

[116] Ion Stoica, Robert 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". In: *IEEE/ACM Transactions on Networking (TON)* 11.1 (2003), pp. 17–32.

[117] *Storj*. `https://storj.io/`. Accessed: 2018-10-27.

[118] Daniel Stutzbach and Reza Rejaie. "Understanding Churn in Peer-to-peer Networks". In: *Proceedings of the Sixth ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil: ACM, 2006, pp. 189–202.

[119] *Symform*. `www.symform.com`. Accessed: 2018-07-15.

[120] Gerard Tel. *Introduction to Distributed Algorithms*. Second. New York, NY, USA: Cambridge University Press, 2001.

[121] Håkan Terelius, Damiano Varagnolo, and Karl Henrik Johansson. "Distributed Size Estimation of Dynamic Anonymous Networks". In: *Proceedings of the 51th IEEE Conference on Decision and Control, CDC 2012, December 10-13, 2012, Maui, HI, USA*. 2012, pp. 5221–5227.

[122] Eli Upfal. "Tolerating a Linear Number of Faults in Networks of Bounded Degree". In: *Information and Computation* 115.2 (1994), pp. 312–320.

[123] Vasileios Vlachos, Stephanos Androutsellis-Theotokis, and Diomidis Spinellis. "Security Applications of Peer-to-peer Networks". In: *Computer Networks* 45.2 (2004), pp. 195–205.

[124] Duncan J. Watts and Steven H. Strogatz. "Collective Dynamics of "Small-world" Networks". In: *Nature* 393 (1998), pp. 440–442.

[125] *Website of Bitcoin*. `https://bitcoin.org/en/`. Accessed: 2018-07-15.

[126] *Website of BitTorrent*. `https://www.bittorrent.com/`. Accessed: 2018-07-15.

[127] Nicholas Wormald. “Models of Random Regular Graphs”. In: *Surveys in Combinatorics, 1999*. Ed. by J. D. Lamb and D. A. Preece. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999, pp. 239–298.

[128] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Tech. rep. University of California at Berkeley, CA, USA, 2001.