Message Efficient Fault-Tolerant Distributed Computations

A thesis submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the degree of
Doctor of Philosophy in Computer Science

Author: Manish Kumar

under the guidance of

Dr. Anisur Rahaman Molla
Indian Statistical Institute Kolkata

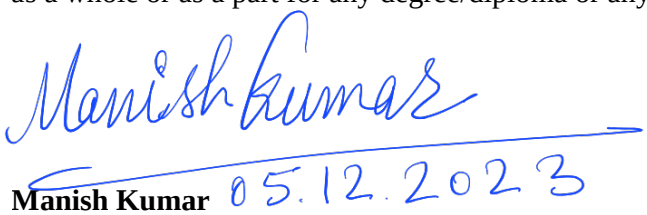
Cryptology and Security Research Unit
Indian Statistical Institute
203 Barrackpore Trunk Road
Kolkata, West Bengal
India - 700 108

November 2023

To the well-wishers; here, there and everywhere!

Declaration of Authorship

I, **Manish Kumar**, a student of Cryptology and Security Research Unit, of the Ph.D. program of Indian Statistical Institute, Kolkata, hereby declare that the investigations presented in this thesis are based on my works and, to the best of my knowledge, the materials contained in this thesis have not previously been published or written by any other person, nor it has been submitted as a whole or as a part for any degree/diploma or any other academic award anywhere before.



05.12.2023

Manish Kumar
Cryptology and Security Research Unit,
Indian Statistical Institute, Kolkata 203 Barrackpore Trunk Road,
Kolkata, West Bengal, India - 700108

Scientists are not stupid but they might be negligent.

Acknowledgements

I would like to take this opportunity to express my heartfelt appreciation to the individuals who have played a pivotal role in shaping my path to success. This remarkable journey has not only shaped my character, career, and disposition but has also provided me with invaluable lessons through a blend of triumphs and challenges. It has encompassed a transformation from attachment to detachment, from moments of frustration to cultivating patience, and from feeling lost to finding my way. This expedition has been marked by the thrilling drama and adventures of tackling research problems, essentially a life within a life. Throughout this exploration, the precise trajectory may linger in uncertainty, but the value of experience, support, and, above all, guidance cannot be underestimated.

I consider myself incredibly fortunate to have thrived under the mentorship of Dr. Anisur Rahaman Molla. I sincerely thank my supervisor, Dr. Molla, for his unwavering support, invaluable advice, constant encouragement, and unyielding motivation. His passion and teaching approach has been immensely inspiring, igniting a strong drive within me. His guidance and extensive knowledge have been instrumental in facilitating my research endeavors and the completion of this thesis. Having a supervisor who has consistently stood by my side is a true privilege, and I will forever remain indebted to him for his priceless guidance.

I am profoundly grateful to Dr. Debrup Chakraborty, my esteemed advisor during my master's (M.Tech.) dissertation. Under his mentorship, I was inspired to delve into unexplored realms of knowledge, as he instilled in me a deep sense of curiosity and enthusiasm. With his patience and uplifting guidance, I wholeheartedly immersed myself in my research pursuits. I would also like to extend my appreciation to Dr. Sushmita Ruj for her guidance during the coursework and initial stage of my Ph.D. Her support and insights were instrumental in laying a strong foundation for my research endeavors.

My Ph.D. thesis would not have been completed without the generous support of my collaborator, for which I owe a deep sense of gratitude to Dr. Sumathi Sivasubramaniam. Her insightful discussions and cooperation were invaluable to my research. I am also immensely grateful for the opportunity to collaborate with an exceptional group of individuals throughout my Ph.D. journey, namely Prof. John Augustine, Dr. Gokarna Sharma, Dr. Sasanka Roy, and Prof. Ajay Kshemkalyani. Their talent and dedication played a pivotal role in shaping my research. I extend my heartfelt appreciation to Dr. Subhra Mazumdar, Prabhat Kumar Chand, Dr. Manish Kumar, Dr. Diptendu Chatterjee, Akanksha Dixit, Nishant Dhanaji Nikam, Serene Rasheed, and Rik Banerjee for their invaluable contributions and thought-provoking discussions, which significantly enriched my work. Furthermore, I consider myself fortunate to have had fruitful interactions with Prof. Gopal Pandurangan and Prof. Costas Busch during conferences, as their insights added tremendous value to my research.

Friends play an indispensable role in life, and I consider myself extremely fortunate to have companions like Joginder, Sandeep, Deepak, Pushpinder, and Indu. Their friendship has added depth to my research experience, creating indelible memories that I will always cherish. The laughter, camaraderie, and shared passion for knowledge across diverse research areas have been a source of joy and growth. Moreover, the friendships I share with Sandeep, Subhra, Bharadwaj, Debasmita, Mohammad, and Sreyosi have brought contentment not only in my research but also in my personal life. Their support and presence have been a constant source of strength during challenging times. I extend my heartfelt appreciation to my friends and seniors at ISI Kolkata for their invaluable contributions to my journey. Special thanks to Prabhat, Diptendu da, Snehal di, Susant da, Ajeet sir, Laltu da, Avishek da, Samir da, Nishant, Prabal da, Akanksha, Sunil, Manabendra, Sanjana di, Debendranath, Sumit, Sayeed, Sarbendu da, Durgesh da, Mallesham da, Pritam da, Santlaal, Abhilash, Yash, Ambar, Nayana di, Mostaf da, Soumya da, Panchalika, Ajay, Bhuvan, Debajyoti, Sayak, Aniruddha da, Subhadip da, Suman, Rakesh, Gourav, Anup, Nirupam, and many others who have generously offered their assistance in various ways at different times. Their unwavering support and companionship have been invaluable, and I am truly grateful for every moment we have shared.

I convey my wholehearted love and deep respect for my family, who have always provided me with a sense of security and shelter in this vast world. Words cannot adequately express my gratitude towards them; I simply want to acknowledge their unwavering love, selfless sacrifices, and remarkable patience throughout my academic journey. My parents, elder brother and sister-in-law, and sister and brother-in-law have consistently exerted a tremendous influence on my life. I express heartfelt gratitude to my niece, Dhavika, and nephew, Namit, for filling my life with joy and preserving a lively atmosphere through their innocence. Furthermore, I extend my gratitude to Brother Braham Prakash, whose presence has served as a steadfast anchor, empowering me to embrace a life of limitless possibilities and unyielding courage.

Finally, I would like to express my gratitude to those individuals who might have been inadvertently omitted but have had an impact on my journey. To every person who has crossed my path and shown me kindness, I offer my heartfelt thanks and appreciation—"*Jis-Jis Se Path Par Sneh Mila, Har Uss Raahi Ka Dhanywaad*" (translated as "To every traveler who has shown me affection along the way, I am grateful").

Date: 17^{th} November 2023 Manish Kumar

Abstract

The thesis focuses on exploring the message complexity of some fundamental problems – leader election, agreement, and graph realization. Leader Election and Agreement problems are widely applicable in various domains such as sensor networks, IoT networks, grid computing, peer-to-peer networks, and cloud computing. Achieving low-cost and scalable leader election and agreement protocols with probabilistic guarantees is often desirable in large-scale distributed networks. Furthermore, the rise of permissionless distributed systems has made it necessary to design protocols that can tolerate an arbitrary number of faulty nodes. On the other hand, graph realization problems deal with constructing graphs that satisfy certain predefined properties (such as a degree sequence) in the presence of crashes. Despite intensive research, there has yet to be a practical solution to fault-tolerant problems for large-scale networks. One key reason for this is the large message complexity of currently known protocols. In this thesis, we focus on two main questions: (1) How efficiently leader election, agreement, and graph realization can be computed in a distributed network? (2) What can be the resilience of the network and how does it affect the complexity?

In this thesis, we study four problems to address the above questions: (i) Leader election and agreement under crash fault (ii) Byzantine agreement (BA) (iii) Distributed graph realization, and (iv) Leader election in diameter-two networks. We present randomized (Monte Carlo) algorithms for leader election and agreement problems that achieve sublinear (in n, number of nodes) message complexity in the implicit version of the two problems when tolerating more than a constant fraction of the faulty nodes. Our algorithms tolerate any number of faulty nodes up to $(n - \text{polylog}\, n)$ which is compensated by the increased complexity. The message complexity (and also the time complexity) of our algorithms is optimal (up to a $\text{polylog}\, n$ factor). Further, we study the message complexity of authenticated Byzantine agreements under an honest majority. We focus on the "implicit" Byzantine agreement problem and show that a sublinear message complexity BA protocol under honest majority is possible in the standard PKI model when the nodes have access to an unbiased global coin and hash function. Our algorithm is optimal (up to a $\text{polylog}\, n$ factor) and works in anonymous networks, where nodes do not know each other. We further study the graph realization problem in the Congested Clique model of distributed computing under crash faults. Our main result is a $O(f)$-round deterministic algorithm for the degree-sequence realization problem in a n-node Congested Clique, of which f nodes could be faulty $(f < n)$. The algorithm uses $O(n^2)$ messages. Our results are optimal in both the models with or without the knowledge of the neighbors (a.k.a. KT_1 and KT_0 model) w.r.t the number of rounds and the messages simultaneously. Later, we investigate the leader election problem in diameter-two networks. We present a $O(\log n)$-round deterministic leader election algorithm which incurs optimal $O(n \log n)$ messages without the knowledge of n.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Leader election and agreement are two essential problems in distributed computing that have been extensively researched for the past four to five decades since their introduction [100, 101, 116]. These problems have gained significant interest due to their direct application in real-world distributed systems, which are susceptible to faults. For example, Akamai, a content delivery network, uses distributed leader election to achieve fault tolerance, and the Paxos agreement protocol employs leader election for consensus [114, 30]. The importance and widespread application of these problems are prevalent in various domains such as sensor networks [128], IoT networks [120], grid computing [8], peer-to-peer networks [99, 121, 124, 125], and cloud computing [135]. In large-scale distributed networks, it is desirable to achieve low-cost and scalable leader election and agreement protocols, even with probabilistic guarantees. Moreover, with the rise of permissionless distributed systems [64], it is also required to design protocols that can tolerate an arbitrary number of faulty nodes. However, despite intensive research, there is still no practical solution to fault-tolerant agreement or leader election for large-scale networks. One of the main reasons for this is the high message complexity of current protocols, which has been a concern in many systems papers [5, 7, 28, 108, 135]. It is desirable to have simple and lightweight protocols that can be easily implemented. On the other hand, graph realization problems deal with constructing graphs that satisfy certain predefined properties (such as a degree sequence) in the presence of crashes. Graph Realization problems have been studied extensively in the literature, mainly in the sequential setting and recently in distributed fault-free setting [12]. This thesis focuses on the following fundamental questions in solving fault-tolerant computation problems in synchronous distributed networks: What is the minimum number of messages required to solve fault-tolerant leader election, agreement and graph realizations?

The problem of leader election involves a group of nodes within a distributed network choosing a single leader among themselves. The elected leader is the only node permitted to output its decision as the leader, while all other nodes output their decision as non-leader. In some applications, the non-leader nodes do not require knowledge of the leader's identity (e.g. IP address), making the implicit variant of leader election sufficient [11, 96, 97, 106]. In contrast, the explicit version necessitates the non-leader nodes' awareness of the leader node's identity number (e.g. IP address). The classical agreement problem has an implicit variant, in which each node starts with an input value (0 or 1). The objective of the agreement problem is to ensure that a non-empty subset of nodes agrees on the same value (consensus condition) and that the common value is the input

value of some nodes (validity condition). In the explicit version, all nodes must agree on the same input value. The implicit agreement problem is a generalization of the explicit agreement problem, first proposed by [14]. For both the problems, leader election and agreement, the message lower bound for the explicit version is trivially $\Omega(n)$.

We are examining the implicit version of the leader election and agreement problems in the context of crash-fault, i.e., a faulty node does not respond after crashing throughout the execution. It is worth noting that any solution to the implicit version of these two problems can be effortlessly extended to solve their explicit versions in $O(1)$ rounds with optimal time and messages (i.e., in $O(1)$ rounds and $O(n)$ messages). The implicit version of the problems is more flexible because all nodes need not agree on the leader or be aware of the agreed value. Therefore, it is possible to achieve agreement (either on a leader or a value) with fewer communications. In particular, one can expect sublinear message bounds, unlike the case of the explicit version of the problems when all nodes need to agree, which requires at least n messages.

In the crash-fault setting, a faulty node may fail by crashing and do not respond throughout the execution. In this setting, our research is closely linked to Gilbert-Kowalski's [59] state-of-the-art work, which explores the message complexity of the agreement problem in the same scenario. It provides an optimal $O(n)$ message complexity algorithm for an explicit agreement that tolerates up to $n/2-1$ faulty nodes. Note that the work of Gilbert-Kowalski [59] assumes that nodes are aware of their neighbors, and if this is not the case (as in our research), the message complexity will be $O(n \log n)$. In contrast, our implicit agreement algorithm can be extended to an explicit agreement algorithm that has a message complexity of $O(n \log n)$ while tolerating any linear fraction of faulty nodes. Additionally, our algorithm is more general in that the message complexity (and time complexity) depends on the number of faulty nodes, allowing for tolerance of more faulty nodes, up to $n-\log^2 n$. Furthermore, our agreement algorithm works in anonymous networks where nodes are unaware of their neighbors and is scalable, meaning it sends only a constant number of bits over an edge per round. We also demonstrate an agreement problem's matching lower bound, which is optimal up to a $\mathrm{polylog}\, n$ factor. Kutten et al. [97] presented an optimal sublinear randomized algorithm for the implicit leader election problem in the *fault-free* setting (i.e., a network with no faulty nodes) and a tight lower bound on the message complexity (up to a $\mathrm{polylog}\, n$ factor). We also provide a non-trivial lower bound on the message complexity, which is optimal for more than a constant fraction of faulty nodes in the network (up to a $\mathrm{polylog}\, n$ factor). Both the problems have $O(\log n)$ round complexity for any constant fraction of faulty nodes, which is almost optimal, due to the lower bound $\Omega(\log n/\log\log n)$ shown in [37]. Our algorithms are lightweight and scalable, as they are only based on sampling and send only $O(\log n)$ bits through an edge per

round. Moreover, our algorithms work in anonymous networks where each node initially does not know the identity of its neighbors.

The other fault-tolerant setting is the Byzantine agreement problem, which is a well-known and extensively studied problem in distributed networks [100, 11, 106]. In this problem, every node in the network is initialized with an input value, and the goal is to achieve the following two conditions: (i) all honest nodes should agree on the same input value, and (ii) if all honest nodes receive the same input value, they must agree on that value. The Byzantine agreement problem must be solved in the presence of a constant fraction of Byzantine nodes, which can behave arbitrarily and deviate from the protocol followed by the honest nodes. Byzantine agreement is an essential building block for creating secure and resilient distributed systems, and it has found widespread and continuous application in numerous domains such as wireless networks [81, 103, 134, 129], sensor networks [128], grid computing [8], peer-to-peer networks [124] and cloud computing [135], cryptocurrencies [26, 44, 3, 80, 109], secure multi-party computation [24], and so on.

King and Saia's [77] work introduced a Byzantine agreement algorithm that overcomes the quadratic message complexity barrier in synchronous, complete networks. Their algorithm has a message complexity of $\tilde{O}(n^{1.5})$. Subsequently, Braud-Santoni et al. [27] improved upon this result, achieving a message complexity of $\tilde{O}(n)$. However, both of these works assume that nodes know the IDs of other nodes beforehand. This model is commonly referred to as the KT_1 model (knowledge till hop 1) [118]. In contrast, the KT_0 model assumes that nodes do not have prior knowledge of their neighbors [118], which is more relevant to modern distributed networks that are permissionless. While nodes in the KT_0 model can easily learn their neighbors by communicating with all of them in a single round, this would incur a message complexity of $\Omega(n^2)$.

Our primary objective is to analyze the message complexity of the Byzantine agreement problem in the KT_0 model, assuming the availability of cryptographic setup and a global coin (generate the shared random bits), as described in [119]. Specifically, we focus on the implicit version of the Byzantine agreement, where only a subset of the honest nodes needs to decide on an input value. Our result introduces the first *sublinear message complexity* Byzantine agreement algorithm and at the same time tolerates optimal resilience, i.e., $f \leqslant (1/2 - \epsilon)n$, where n and f are the number of nodes and number of Byzantine nodes in the network, and $\epsilon > 0$ is a fixed constant. Our algorithm is straightforward and easy to implement, making it highly desirable for practical applications.

In the crash-fault setting, we explore another problem known as the graph realization problem in a complete network. In general, graph realization problems deal with constructing graphs that satisfy certain predefined properties (such as a degree sequence). The area was mostly focused on realizing graphs with specified degrees [68] while other properties such as connectivity [52, 50, 51],

flow [62] and eccentricities [23, 102] have also been studied. The *degree-sequence* realization problem has been explored widely in the centralized setting. Typically, the problem consists of a sequence of non-negative numbers $\mathcal{D} = (d_1, d_2, \ldots, d_n)$, the degree-sequence problem asks if $\mathcal{D}$ is *realizable*. A degree sequence $\mathcal{D}$ is realizable if there is a possibility of constructing a graph of n nodes with the given degree sequence. These degree sequences are provided to the nodes, one for each node.

Firstly, we solve the graph realization problem in the KT_1 (knowledge till hop 1) model and provide the matching lower bound. Then we also solve the problem in the KT_0 (knowledge till hop 1) model and achieve results similar to KT_1 model. In both settings, we achieve the graph realization in $O(f)$ rounds and $O(n^2)$ messages in which f nodes could be faulty ($f < n$). The algorithms are deterministic for the degree-sequence realization problem in a n-node Congested Clique and the value of the f is unknown. Notice that KT_1 model is the stronger model (having more assumptions) than KT_0. Therefore, KT_1 model's lower bound is the trivial lower bound for the KT_0 model while KT_0 model algorithm also works in the KT_1 model. As a by product, our approach also solves the information spreading problem where each node knows the value of every other node.

We also investigate the leader election problem in a non-faulty setting in diameter-two networks. The problem of leader election has been extensively studied in the literature in fault-free settings with a focus on both message and round complexity for various graph structures, including rings [101, 136], complete graphs [4, 14, 69, 83, 85, 86, 130], and diameter-two networks [31]. Additionally, leader election has been studied in general graphs [54, 60, 97, 106, 117]. Previous works have primarily focused on providing deterministic solutions, but randomized algorithms have also been explored, mainly to reduce the message complexity [14, 60, 96, 97] and their references. Kutten et al. established the fundamental lower bound for leader election in general graphs with $\Omega(m)$ message complexity and $\Omega(D)$ round complexity [96], where m and D are the number of edges and diameter of the graph, respectively. This bound applies to all graphs with diameters greater than two, whether the algorithm is deterministic or randomized. For the clique, Kutten et al. recently established a tight message lower bound of $\Omega(n \log n)$ for deterministic algorithms under simultaneous wake-up of the nodes [98]. This result was previously shown by Afek and Gafni in 1991 [4], assuming adversarial wake-up. In 2020, a deterministic algorithm with $O(n \log n)$ message complexity was proposed for diameter-two networks by Chatterjee et al. [31].

Our research is closely related to the work conducted by Chatterjee et al. [31], who investigated leader election in diameter-two networks, specifically the implicit version. The authors presented a deterministic algorithm with a message complexity of $O(n \log n)$ and a round com-

plexity of $O(\log n)$. However, their algorithm required prior knowledge of the network size, n. In contrast, our algorithm can elect a leader explicitly without prior knowledge of n. Our algorithm has a message complexity of $O(n \log \Delta)$ and completes in $O(\log \Delta)$ rounds, where Δ is the maximum degree of the graph. Furthermore, we demonstrate how to utilize the edges used during the leader election protocol to construct a broadcast tree for diameter-two graphs, with a message complexity of $O(n \log \Delta)$ and a round complexity of $O(\log \Delta)$. Efficiently computing a broadcast tree is another fundamental problem in distributed computing, and a broadcast tree can serve as a subroutine for many distributed algorithms that seek message efficiency. Chatterjee et al. [31] left open the task of finding a deterministic $\tilde{O}(n)$-message[1] and $\tilde{O}(1)$-round broadcast algorithm in diameter-two networks, which we addressed in this thesis.

The remaining sections of this chapter are structured as follows. We begin in Section 1.1 by introducing an overview of all the notation and definitions which are used throughout the thesis. Section 1.2 presents the distributed computing model that serves as the foundation for our analysis. Section 1.3 provides a brief literature review of the work done in the thesis. In Section 1.4, we outline the contributions of the thesis and its overall organization. Finally, in Section 1.5, we present a list of the publications that were used in the development of this thesis. Furthermore, in each chapter, we provide a detailed explanation of the notation, definitions, and distributed computation model used. We also specify the source of the material, i.e., references to the relevant publications or manuscripts that contributed to the thesis.

1.1 Preliminaries

We introduce fundamental notations and definitions that recur frequently in the thesis.

We follow the standard notation for representing graphs as follows: $G = (V, E)$, where V denotes the set of vertices and E denotes the set of edges. The terms "graph" and "network", "node" and "vertex", and "edge" and "communication link" are used interchangeably. The number of nodes in the graph is represented by n, which is equivalent to the cardinality of V. Similarly, m represents the number of edges or the cardinality of E.

In a fault-tolerant set-up, f represents the number of faults. The fault-tolerant leader election requires electing a leader node from the set of n nodes such that the leader is non-faulty with probability $(1 - f/n)$. We have two types of leader election and agreement – Implicit and Explicit. In implicit, a subset of non-faulty nodes is aware of the leader and agreement. On the other hand,

[1] $\tilde{O}$ hides the $\operatorname{polylog} n$ factor.

in explicit, all the non-faulty nodes are aware of the decision (leader election or agreement value). In the leader election, we elect a node as a leader, which is represented by its ID. In agreement, it is the input value provided to the nodes based on which nodes agree. These input values can be binary (0 or 1) or arbitrary based on the problem statement. In the case of graph realization problem, the problem consists of a sequence of non-negative numbers $\mathcal{D} = (d_1, d_2, \ldots, d_n)$, the degree-sequence problem asks if $\mathcal{D}$ is *realizable*. A sequence $\mathcal{D}$ is said to be *realizable* if there is a graph of n nodes whose sequence of degrees matches $\mathcal{D}$.

In this thesis, we frequently employ the $\tilde{O}$ notation, which conceals the $\operatorname{polylog} n$ factor. Essentially, $\tilde{O}(f)$ denotes $O(f \log^c n)$ for a constant c. Additionally, we commonly use the abbreviated phrase "w.h.p." or "with high probability," signifying a probability of at least $1-1/n^{\epsilon}$ for a constant ϵ, where n corresponds to the number of nodes present in the network.

1.2 Distributed Computing Model and Definitions

We consider the distributed network for a graph $G = (V, E)$ of n nodes. Unless specified otherwise, throughout the thesis we restrict our attention to undirected graphs in which communication is bidirectional, and unweighted graphs where all the edges are the same without any further cost. Our graph is fully connected, i.e., every node is directly connected to all other nodes except Chapter 5, where we consider the graph with diameter-two in which a node is at most two hop away from any other node. We assume that nodes do not know the value of n. However, a node knows the value of n simply by its number of ports connected to the other nodes when the network is complete. But in the case of a diameter-two network (Chapter 5), nodes are unaware of the value of n. Initially, each node has limited knowledge of the neighboring nodes and each node is assigned an identity number in a deterministic setting (in Chapter 4 and 5). In a randomized setting (in Chapter 2 and 3), nodes select their identity number randomly from a large sample set such that all the nodes have different identity numbers. Nodes may know the identity number (e.g. IP address) of the neighboring nodes based on the problem specification. Each node has a unique identity number of size $O(\log n)$ bits. The model is known as *clean network model* in the sense that the nodes are unaware of their neighbors' identity numbers initially, also known as KT_0 (knowledge till 0 hop) model [118]. Except for Chapter 4 (in Section 4.4 and Section 4.5) where we assume additional communications between the nodes, in which, nodes can directly communicate to other nodes if they know their identity number (e.g., IP address). We call this KT_1 (knowledge till 1 hop) model and define it as where nodes know the identity number of the other nodes and the port connecting to the identity numbers.

Our focus is on the message-passing model of distributed computing. The model involves nodes communicating by exchanging messages in rounds, where a node can communicate with any neighbor by sending a message. The network operates synchronously, with nodes running at the same speed and messages taking exactly one round to arrive at their intended recipient, i.e., a message sent in the r^{th} round would be received by $(r+1)^{th}$ round. At the beginning of execution, all nodes start simultaneously and keep track of the round number. The round complexity of an algorithm is the number of rounds taken by an algorithm from the start until the termination. In the case of messages' size to ensure the scalability, we consider *CONGEST* communication model [118], where a node is allowed to send a message of size $O(\log n)$ or $O(\text{polylog}(n))$ bits through an edge per round. This addresses the real-world computer problem and helps to keep the bandwidth small. The message complexity of an algorithm is the total number of messages sent by all the nodes throughout the execution of the algorithm. On the other hand, there exists another distributed computing model where message size (bandwidth restriction) is relaxed. That model is *LOCAL* communication model, where nodes are allowed to send a message of arbitrary size (polynomial in n) through an edge per round. We avoid using the LOCAL model during the problem-solving phase but show some lower bound results which are even feasible in the presence of this model.

In the case of fault-tolerant setup, a network is called f-resilient if at most f nodes may be faulty. We consider two types of fault – crash-fault and Byzantine fault. In the crash-fault (Chapter 2 and 4), we assume that an arbitrary subset of the nodes in the clique of size up to $f < n$ may fail by crashing. A faulty node may crash in any round during the execution of the algorithm. If a node crashes in a round, then an arbitrary subset (possibly all) of its messages for that round may be lost as determined by an adversary and may not be reached the destination. The crashed node halts (inactive) in the further rounds of the algorithm. If a node does not crash in a round, then all the messages that it sends in that round are delivered to the destination [11]. On the flip side, a Byzantine faulty node (Chapter 3) can behave maliciously such that it sends any arbitrary message or no message in any round to mislead the protocol, e.g., it may send different input values to different nodes, or it may not send any message to some of the nodes in a particular round.

Both of the aforementioned settings consider two types of adversaries: static (also known as non-adaptive) and adaptive. In Chapters 2 and 3, we assume a *static* adversary controls the faulty nodes, selecting them before the algorithm's execution begins. Conversely, in Chapter 4, an adaptive adversary can determine which nodes become faulty during the algorithm's execution. Also, the adversary chooses when and how a node behaves, which is unknown to the nodes. Further, the adversary is rushing and has full information – the adversary knows the states of all the nodes

and can view all the messages in a round before sending out its own messages for that round. In randomized algorithms (in Chapter 2 and Chapter 3), each node has access to an arbitrary number of private random bits.

Efficiency measurement for a distributed algorithm can involve different parameters, such as the number of rounds, messages sent, and more. However, this thesis focuses on minimizing the number of messages while keeping the number of rounds as low as possible. It is worth noting that local computation performed by a node is considered "free" and does not impact our efficiency measures. However, we do perform polynomial-cost computations (in terms of time and space) on nodes locally.

One approach to designing a message-efficient algorithm may be to wait for a specific event without sending any messages, but this can be time-consuming in terms of the number of rounds. Therefore, our goal is to construct distributed algorithms that are both message-efficient and have low round complexity. Throughout this thesis, we present message-efficient algorithms from Chapter 2 to Chapter 5 that are almost round-optimal (within $\mathrm{polylog}\, n$) in their respective settings. Specifically, in Chapter 4, we present algorithms that are optimal in terms of both message and round complexity, with the lower and upper bounds of the algorithm being exactly the same.

Let us formally define the main results' terminology that are used in the thesis.

Definition 1.1 (Fault-Tolerant Leader Election, Chapter 2)**.** *Consider a complete network of n nodes, in which at most f nodes could be faulty and the remaining $(n - f)$ nodes are non-faulty. The fault-tolerant leader election requires electing a leader node from the set of n nodes such that the leader is non-faulty with probability $(1 - f/n)$.*

Definition 1.2 (Fault-Tolerant Agreement, Chapter 2)**.** *Consider a complete network of n nodes, in which at most f nodes could be faulty and the remaining $(n - f)$ nodes are non-faulty. Suppose initially each node has an input value in $\{0, 1\}$. An implicit agreement holds when the final state is either $\{0, \perp\}$ or $\{1, \perp\}$ and at least one node has a state other than $\perp$ (which should be the input value of some node), where $\perp$ denotes the 'undecided' state. In other words, all the decided nodes must agree on the same value which is an initial input value of some node and there must be at least one decided node in the network.*

Definition 1.3 (Implicit Byzantine Agreement, Chapter 3)**.** *Suppose initially all the nodes have an input value (say, provided by an adversary). An implicit Byzantine agreement holds when the following properties hold: (i) the final state of all the non-Byzantine nodes is either "decided" or "undecided"; (ii) all the "decided" non-Byzantine nodes must agree on the same value (consistency property); (iii) if all the non-Byzantine nodes have the same input value then they must*

decide on that value (validity property); (iv) all the non-Byzantine nodes eventually reach to the final state, where at least one non-Byzantine node must be in the "decided" state (termination).

Definition 1.4 (Distributed Graph Realization with Faults, Chapter 4)**.** *Let $V = \{v_1, \dots v_n\}$ be the set of nodes in the network and $\mathcal{D} = (d_1, d_2, \dots, d_n)$ be an input degree sequence such that each d_i is only known to the corresponding node v_i. Let $F \subset V$ be an arbitrary subset of faulty nodes in the network, such that $|F| = f \leq n - 1$. Let us define $\mathcal{D}' \subseteq \mathcal{D}$ be the modified degree sequence after losing the degrees of some faulty nodes and G' be the corresponding overlay graph over $\mathcal{D}'$. The distributed degree realization with faults problem requires that the* non-faulty *nodes in V construct a graph realization of $\mathcal{D}'$ such that in the resulting overlay graph G', the following conditions hold:*

1. *$|\mathcal{D}'| \geq n - |F|$.*
2. *$\mathcal{D} - \mathcal{D}'$ is the degree sequence corresponding to some nodes that crashed.*
3. *For any edge $e = (u, v) \in G'$, either u or v (or both) knows of the existence of e.*

The required output is an overlay graph G' if $\mathcal{D}'$ is realizable; otherwise, the output is "unrealizable".

Definition 1.5 (Implicit Leader Election, Chapter 5)**.** *Consider an n-node distributed network. Let each node maintain a state variable that can be set to a value in $\{\perp, NONELECTED, ELECTED\}$, where $\perp$ denotes the 'undecided' state. Initially, all nodes set their state to $\perp$. In the implicit version of leader election, it requires that exactly one node has its state variable set to $ELECTED$ and all other nodes are in state $NONELECTED$. The unique node whose state is $ELECTED$ is the leader.*

Definition 1.6 (Explicit Leader Election, Chapter 5)**.** *Consider an n-node distributed network. Let each node maintain a state variable that can be set to a value in $\{\perp, NONELECTED, ELECTED\}$, where $\perp$ denotes the 'undecided' state. Initially, all nodes set their state to $\perp$. In the explicit version of leader election, it requires that exactly one node has its state variable set to $ELECTED$ and all other nodes are in state $NONELECTED$. Further, the $NONELECTED$ nodes must know the identity of the node, whose state is $ELECTED$, the leader.*

1.3 Literature Review

We state the brief literature review of the thesis chapter-wise. Later, we provide a detailed description of the related work in their respective chapter.

In the first contributory chapter (Chapter 2) we discuss the leader election and agreement problem in faulty setup. Initially, the leader election was studied by Le Lann [101] in ring networks and then by the seminal work of Gallager-Humblet-Spira [54] in general graphs. The problem is well studied in the complete network itself [4, 69, 83, 85, 86, 130], and reference therein. Similarly, the agreement is also extensively studied in the class of complete networks [116, 106, 59, 37, 32, 33, 34, 36]. The most closely related work to ours, in a fault-free setting, is by Kutten et al. [97] which studied the (implicit) leader election problem in complete networks and showed sub-linear message bounds using randomization. They also showed an almost matching lower bound for randomized leader election. Later, Augustine et al. [14] introduced and studied implicit agreement problem in fault-free settings and showed similar sub-linear bounds for agreement. Our work is inspired by the works of [97, 14, 59], but in the faulty setting. In the crash-fault setting, Gilbert-Kowalski [59] studied the message complexity of the agreement. They presented an optimal $\mathcal{O}(n)$ message complexity (explicit) agreement protocol which tolerates at most $n/2-1$ faulty nodes in the KT_1 model. The time complexity is $\mathcal{O}(\log n)$. While their algorithm solves the implicit agreement in a smaller set of nodes (of size $\mathcal{O}(\log n)$) in the course of the explicit agreement, they did not mention the implicit version of the agreement. Their implicit agreement solution uses $\tilde{\mathcal{O}}(n^{1/2})$ messages ($\tilde{\mathcal{O}}$ hides a $\operatorname{polylog} n$ factor) and $\mathcal{O}(\log n)$ rounds. In comparison, our agreement algorithm matches all these bounds while tolerating more faulty nodes. Table 2.1 summarizes the state-of-the-art in the crash-fault setting.

The second contributory chapter (Chapter 3) discusses the implicit Byzantine agreement. Byzantine agreement was introduced by Lamport, Shostak, and Pease [100, 116]. They presented protocols and fault tolerance bounds for two settings (both synchronous). Without cryptographic assumptions (the unauthenticated setting), Byzantine broadcast and agreement can be solved if $f < n/3$. Assuming digital signatures (the authenticated setting), Byzantine broadcast can be solved if $f < n$ and Byzantine agreement can be solved if $f < n/2$. The initial protocols had exponential message complexities [100, 116]. Fully polynomial protocols were later shown for both the authenticated $(f < n/2)$ [40] and the unauthenticated $(f < n/3)$ [55] settings. Both protocols require $f + 1$ rounds of communication, which matches the lower bound on round complexity for deterministic protocols [48]. The previous results in the same direction are summarized in Table 3.1.

The third contributory chapter (Chapter 4) discusses the fault-tolerant graph realization problem. Fault-tolerant computation becoming more popular with the prevalence of P2P networks that encourage high decentralization. Often the focus of such research is on maintaining connectivity [16], recovery, or ensuring that the network can tolerate a certain number of faults [133]. In

our work, we focus mainly on ensuring that all the (non-faulty) nodes have the same view of the information in spite of the presence of numerous faults. Graph realization problems have been well studied in the literature, focusing on problems such as realizing graphs with specified degrees [68] as well as other properties, like connectivity and flow [62, 52, 50, 51] or eccentricities [23, 102]. Bar-Noy et al. explored the graph realizations in various settings like vertex-weighted, distance set, and relaxed and approximate graph realizations [19, 18, 20]. Arikati and Maheshwari [9] provide an efficient technique to realize degree sequences in the PRAM model, and in [12], the authors explored graph realization from a distributed setting. However, both of these works assumed a fault-free setting. Here we extend the model to a faulty setting. In [12], Augustine et al. discussed distributed graph realizations on a path (both implicit and explicit). However, in their work, a node is only required to learn its neighbors in the final realization, in our work, the nodes are aware of the entire graph.

The fourth (and last) contributory chapter (Chapter 5) discusses the deterministic leader election in diameter-two network. In the deterministic case, it is known that $\Theta(n \log n)$ is tightly bound on the message complexity for complete graphs [4, 98]. This tight bound also carries over to the general case as seen from [4, 84, 86]. In our work, we restrict our model to graphs of diameter-two. For diameter-two graphs, Chatterjee and colleagues provide a $O(\log n)$ round algorithm that uses $O(n \log n)$ messages. However, their algorithm requires knowledge of n, our algorithm provides an algorithm that requires no prior knowledge of n and runs in $O(\log n)$ rounds with $O(n \log n)$ message complexity.

1.4 Our Contribution and Organization of the Thesis

Within this section, we outline the issues that the thesis tackles, present a summary of the outcomes, and delineate the thesis's structure. The rest of the chapters of the thesis are organized as follows.

Chapter 2 ***Fault-Tolerant Leader Election and Agreement Problems:*** We study a synchronous and fully-connected distributed network of n nodes in the presence of crash-fault nodes. In that, we investigate the message complexity of two problems, leader election, and agreement. In the leader election problem, we have a group of n nodes, and a unique node is elected as a leader. Depending on the nodes' knowledge of the leader, there are two popular versions. In the first version (known as the *implicit* leader election), the non-leader nodes are not required to know the leader's identity; it is enough for them to know that they are not the leader. In the other version (known as *explicit* leader election), the non-leader nodes are required to learn

the leader's identity. The implicit version of the leader election is the generalized version of the (explicit) leader election. Similarly, we define the implicit and explicit agreement problems. In the agreement problem, each node has some input value, and eventually, a non-empty subset agrees on some given value based on the version of the agreement (implicit or explicit).

We present randomized (Monte Carlo) algorithms for both problems and also show non-trivial lower bounds on the message complexity. Our algorithms achieve sublinear message complexity in the so-called implicit version of the two problems when tolerating more than a constant fraction of the faulty nodes. These algorithms consider the *CONGEST* communication model [118], where a node is allowed to send a message of size $O(\log n)$ bits through an edge per round. The message complexity of an algorithm is the total number of messages sent by all the nodes throughout the execution of the algorithm. On the other hand, our lower bounds even hold in the *LOCAL* communication model, where nodes are allowed to send a message of arbitrary size through an edge per round.

In comparison to the state-of-art, our results improved and extended the works of Gilbert-Kowalski [59] (which studied only the agreement problem) in several directions. Specifically, our algorithms tolerate any number of faulty nodes up to $(n - \text{polylog}\, n)$. The message complexity (and also the time complexity) of our algorithms is optimal (up to a $\text{polylog}\, n$ factor). Further, our algorithm works in anonymous networks, where nodes do not know each other. This chapter is based on the joint work with Anisur Rahaman Molla [92].

Chapter 3 ***Authenticated Byzantine Agreement Problem:*** We study the message complexity of authenticated Byzantine agreement (BA) in synchronous, fully-connected distributed networks under honest majority. We focus on the so-called *implicit* Byzantine agreement problem where each node starts with an input value and at the end a non-empty subset of the honest nodes should agree on a common input value by satisfying the BA properties (i.e., there can be undecided nodes)[2]. The Byzantine agreement problem is required to satisfy: (i) the honest nodes must decide on the same input value; and (ii) if all the honest nodes receive the same input value, then they must decide on that value[3].

We show that a sublinear (in n, number of nodes) message complexity BA protocol under honest majority is possible in the standard PKI model when the nodes have access to an unbiased global coin and hash function. In particular, we present a randomized Byzantine

[2]Implicit BA is a generalization of the classical BA problem. Throughout, we write Byzantine agreement or BA to mean implicit Byzantine agreement.

[3]Throughout, we interchangeably use the term 'non-Byzantine' and 'honest', and similarly, 'Byzantine' and 'faulty'.

agreement algorithm which, with high probability achieves implicit agreement, uses $\tilde{O}(\sqrt{n})$ messages and runs in $\tilde{O}(1)$ rounds while tolerating $(1/2-\epsilon)n$ Byzantine nodes for any fixed $\epsilon > 0$, the notation $\tilde{O}$ hides a $O(\text{polylog}\, n)$ factor. The algorithm requires a standard cryptographic setup PKI and hash function with a static Byzantine adversary. The algorithm works in the CONGEST model and each node does not need to know the identity of its neighbors, i.e., works in the KT_0 model. The message complexity (and also the time complexity) of our algorithm is optimal up to a $\text{polylog}\, n$ factor, as we show a $\Omega(\sqrt{n})$ lower bound on the message complexity. We also perform experimental evaluations and highlight the effectiveness and efficiency of our algorithm. The experimental results outperform the theoretical guarantees. We further extend the result to Byzantine subset agreement, where a non-empty subset of nodes should agree on a common value. We analyze several relevant results which follow from the construction of the main result.

To the best of our knowledge, this is the first sublinear message complexity result of the Byzantine agreement. A quadratic message lower bound is known for any deterministic BA protocol (due to Dolev-Reischuk [JACM 1985]). The existing randomized BA protocols have at least quadratic message complexity in the honest majority setting. Our result shows the power of a global coin in achieving significant improvement over the existing results. It can be viewed as a step towards understanding the message complexity of randomized BA in distributed networks with PKI and global coin. This chapter is based on the joint work with Anisur Rahaman Molla. The manuscript is submitted to *International Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS 2023).

Chapter 4 ***Fault-Tolerant Graph Realizations Problem:*** In this chapter, we study the graph realization problem in the Congested Clique model of distributed computing under crash faults. We consider *degree-sequence realization*, in which each node v is associated with a degree value $d(v)$, and the resulting degree sequence is realizable if it is possible to construct an overlay network with the given degrees. Our main result is a $O(f)$-round deterministic algorithm for the degree-sequence realization problem in a n-node Congested Clique, of which f nodes could be faulty ($f < n$). The algorithm uses $O(n^2)$ messages. We complement the result with lower bounds to show that the algorithm is tight w.r.t the number of rounds and the messages simultaneously. We also extend our result to the Node Capacitated Clique (NCC) model, where each node is restricted to sending and receiving at most $O(\log n)$ messages per round. In the NCC model, our algorithm solves degree-sequence realization in $O(nf/\log n)$ rounds and $O(n^2)$ messages. These algorithms work for KT_1 (Knowledge Till 1 hop) model where nodes know their neighbors' IDs.

We further study the graph realization problem in the more robust model that is KT_0 (Knowledge Till 0 hop) model, in which each node knows the IDs of all the nodes in the clique, but does not know which port is connecting to which node-ID. We extend the result to KT_0 when the network is anonymous, i.e., the IDs of the neighboring nodes are unknown. We present an algorithm that solves the graph realization problem in the KT_0 model with matching performance guarantees as in the KT_1 model. This chapter is based on the work [91] and the joint work with Anisur Rahaman Molla and Sumathi Sivasubramaniam [94].

Chapter 5 ***Deterministic Leader Election Problem in Diameter-Two Networks:*** We investigate the leader election problem in diameter-two networks. In 2020, Chatterjee et al. [31] studied the leader election in diameter-two networks. They presented a $O(\log n)$-round deterministic implicit leader election algorithm which incurs optimal $O(n \log n)$ messages, but a drawback of their algorithm is that it requires knowledge of n. An important question – whether it is possible to remove the assumption on the knowledge of n was left open in their paper. Another interesting open question raised in their paper is whether *explicit* leader election can be solved in $\tilde{O}(n)$ messages deterministically. In this chapter, we give an affirmative answer to them. Further, we solve the *broadcast problem*, another fundamental problem in distributed computing, deterministically in diameter-two networks with $\tilde{O}(n)$ messages and $\tilde{O}(1)$ rounds without the knowledge of n. In fact, we address all the open questions raised by Chatterjee et al. for the deterministic leader election problem in diameter-two networks.

In our study, we introduce a deterministic algorithm for leader election, specifically designed as an "explicit" approach. This algorithm operates within $O(\log \Delta)$ rounds and requires $O(n \log \Delta)$ messages, where n represents the number of nodes, and Δ denotes the maximum degree of the network. Importantly, our algorithm achieves these complexities without relying on prior knowledge of n. The tightness of the message bound is established through a corresponding lower bound proven by Chatterjee et al. in [31]. Furthermore, we tackle the "broadcast" problem and demonstrate that it can be deterministically solved in $O(\log \Delta)$ rounds using $O(n \log \Delta)$ messages. Specifically, we illustrate that a broadcast tree with a maximum depth of $O(\log \Delta)$ can be computed with the same complexities. This chapter is based on the joint work with Anisur Rahaman Molla and Sumathi Sivasubramaniam [95].

Chapter 6 **Conclusion and Future Work:** We summarize the thesis with its main contribution as well as some interesting problems for future study.

1.5 List of Publications

1. Manish Kumar, Anisur Rahaman Molla. *On the Message Complexity of Fault-Tolerant Computation: Leader Election and Agreement*.
   - In IEEE Transactions on Parallel and Distributed Systems (TPDS), pp. 1115-1127, Volume: 34, Issue: 4, 01 April 2023. DOI: 10.1109/TPDS.2023.3239993.
   - In the Proceedings of 40th ACM Symposium on Principles of Distributed Computing (PODC), Pages 259–262 (Brief Announcement – Invited), virtual event (Originally held in Italy), July 26-30, 2021. DOI: 10.1145/3465084.3467949.
   - In Proceeding of the 24th International Conference on Distributed Computing and Networking, (ICDCN), Pages 294–295 (Doctoral Symposium), Kharagpur, India, January 4-7, 2023. DOI: 10.1145/3571306.3571424.
   - In Proceeding of the 36th IEEE International Parallel & Distributed Processing Symposium, (IPDPS), Page 1283 (Poster at PhD Forum), virtual event (Originally held in France), May 30 – June 3, 2022. IPDPS 2022 PhD Forum.
2. Manish Kumar, Anisur Rahaman Molla. *Authenticated Implicit Byzantine Agreement with Sublinear Message Bounds*. To be published in the Proceedings of 25th International Conference on Distributed Computing and Networking, ICDCN 2024, 4th - 7th January 2024, Chennai, India. DOI: 10.1145/3631461.3631548.
3. Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. *Fault-tolerant graph realizations in the congested clique*. In the Proceedings of 18th International Symposium on Algorithmics of Wireless Networks, ALGOSENSORS 2022, Potsdam, Germany, September 8-9, 2022, volume 13707 of Lecture Notes in Computer Science, pages 108–122. Springer, 2022. DOI: 10.1007/978-3-031-22050-08.
4. Manish Kumar. *Fault-tolerant graph realizations in the congested clique, revisited*. In the Proceedings of 19th International Conference Distributed Computing and Intelligent Technology, ICDCIT 2023, pages 84–97, Springer Nature Switzerland Cham, 2023. DOI: 10.1007/978-3-031-24848-16.
5. Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. *Improved deterministic leader election in diameter-two networks*. In the Proceedings of 13th International Conference on Algorithms and Complexity (CIAC 2023), Larnaca, Cyprus, June 13-16, 2023, Proceedings, volume 13898 of Lecture Notes in Computer Science, page 323-335, Springer, 2023. DOI: 10.1007/978-3-031-30448-4-23.

Chapter 2

Message Efficient Algorithms for Leader Election and Agreement under Crash Fault

This chapter focuses on examining the message complexity of two basic problems – leader election and agreement, in a fully-connected distributed network with crash-fault synchronous[1]. We provide randomized (Monte Carlo) algorithms for both problems and establish non-trivial lower bounds on the message complexity. Moreover, our algorithms achieve sublinear message complexity in the "implicit" version of the problems, when tolerating more than a constant fraction of faulty nodes.

We first present a randomized leader election algorithm that elects a leader with high probability in a complete network with n nodes, in which at least $\lceil \alpha n \rceil$ nodes are non-faulty and the remaining can be faulty, where $\alpha \geq (\log^2 n)/n$. The time complexity of the algorithm is $O(\log n/\alpha)$ rounds and message complexity is $O\left((n^{0.5}\log^{2.5} n)/\alpha^{2.5}\right)$ with high probability. Then we provide a non-trivial lower bound of $\Omega\left(n^{0.5}/\alpha^{1.5}\right)$ messages for any leader election algorithm that tolerates at most $(1-\alpha)$-fraction faulty nodes and succeeds with high probability. This bound holds regardless of the number of rounds used and applies to both *LOCAL* and *CONGEST* models of distributed networks.

Furthermore, we develop a randomized algorithm, tolerating at most $\lfloor (1-\alpha)n \rfloor$ faulty nodes, solves agreement in $O(\log n/\alpha)$ rounds and with high probability and uses only $O\left((n^{0.5}\log^{1.5} n)/\alpha^{1.5}\right)$ messages. And also show a matching lower bound (up to a $\operatorname{polylog} n$ factor) of $\Omega\left(n^{0.5}/\alpha^{1.5}\right)$ messages for any agreement algorithm that tolerates at most $(1-\alpha)$-fraction faulty nodes and succeeds with high probability. This bound also holds regardless of the number of rounds used and applies to both *LOCAL* and *CONGEST* models.

Compared to the state-of-the-art, our research constitutes a significant advancement and broadening of the current literature. Specifically, we have enhanced and expanded the work of Gilbert-Kowalski [59], which concentrated exclusively on the agreement problem, in several ways. Our algorithms can withstand any number of faulty nodes up to $(n - \operatorname{polylog} n)$, and their message and time complexity is optimal (with a $\operatorname{polylog} n$ factor). Our findings also reveal a surprising fact: the

[1]This chapter is based on joint work with Anisur Rahaman Molla (which appeared in IEEE Transactions on Parallel and Distributed Systems 2023) and contains material from [92].

message complexity of leader election and agreement is asymptotically equivalent to that of the fault-free network [97, 14], with a message complexity of $\tilde{O}(n^{1/2})$ (where $\tilde{O}$ conceals a $\mathrm{polylog}\, n$ factor) even for a constant fraction of faulty nodes. Moreover, our lower bounds align with these outcomes.

2.1 Introduction

Leader election and agreement are two fundamental problems in distributed computing, which, along with their variants, have been extensively studied for the last four decades, since their introduction [100, 101, 116]. Over time, the problems gained significant interest due to their direct applications in real-world distributed systems, which are prone to be faulty. For example, the content delivery network Akamai [114] uses distributed leader election as a subroutine to achieve fault-tolerance, Paxos [30] agreement protocol uses leader election for consensus. The importance and widespread application of the two problems are prevalent in many domains, e.g., sensor networks [128], IoT networks [120], grid computing [8], peer-to-peer networks [99, 121, 124, 125] and cloud computing [135]. Often, in large-scale distributed networks, it is desirable to achieve low-cost and scalable leader election and agreement protocols, even with probabilistic guarantees. Furthermore, due to the rise of permissionless distributed systems [64] (where participants can join the system anonymously with virtual IDs without any control over the numbers), it is also desirable to design protocols that may tolerate an arbitrary number of faulty nodes. However, despite intensive research, there has still not been a practical solution to the fault-tolerant agreement or leader election for large-scale networks. One key reason for this is the large message complexity of currently known protocols– a concern which has been raised in many systems papers [5, 7, 28, 108, 135], and at the same time it is desirable to have simple and lightweight protocols that can be implemented easily. In this chapter, motivated by the need for a practical fault-tolerant protocol, we focus on the following fundamental questions in solving fault-tolerant agreement (also known as *consensus*) and leader election in synchronous distributed networks: What is the minimum number of messages required to solve fault-tolerant leader election and agreement?

The leader election problem requires a group of nodes in a distributed network to elect a unique leader among themselves, such that exactly one node must output its decision as the leader, and all the other nodes output their decision as non-leader. These non-leader nodes need not be aware of the identity of the leader. This implicit variant of leader election is quite well-known [11, 96, 97, 106] and is sufficient in many applications. In the explicit version, the non-leader nodes must know the ID of the leader node. Similarly, in the implicit version of the classical (binary)

agreement problem, each node starts with an input value (0 or 1) and the agreement problem requires guaranteeing the following two conditions: (1) A non-empty subset of nodes should agree (or decide) on the same value (consensus condition) and (2) the common value should be the input value of some nodes (validity condition). In the explicit version, all the nodes must agree on the same input value. The implicit agreement, which is a generalization of the (explicit) agreement problem, was first introduced by [14]. It is trivial that $\Omega(n)$ is a message lower bound for the explicit version of both the problems.

We study the implicit version of leader election and agreement under crash-fault. Note that any solution of the implicit version of two problems can be easily extended to solve the explicit versions in $O(1)$ rounds. The flexibility of the implicit version of the two problems lies in the fact that all the nodes need not be agreed on the leader or do not need to know the agreed value. Therefore, in principle, the agreement (either on a leader or a value) may be fulfilled with less communication. In particular, one can expect sublinear message bounds, unlike the case of the explicit version of the problems when all nodes need to agree, which requires at least n messages.

Our work is closely related to the state-of-art Gilbert-Kowalski [59], which studies the message complexity of the agreement problem in the same setting. It presents an optimal $O(n)$ message complexity algorithm for the explicit agreement and tolerates at most $n/2 - 1$ faulty nodes. Note that the algorithm in [59] assumes the nodes know their neighbors; if not known (like ours), the message complexity would be $O(n \log n)$. In comparison to this, our implicit agreement algorithm can be extended to an explicit agreement algorithm that has message complexity $O(n \log n)$ when tolerating any linear fraction faulty nodes. In fact, our algorithm is more general in the sense that the message complexity (and also the time complexity) depends on the number of faulty nodes, and thus, tolerating more faulty nodes, up to $n - \log^2 n$. Furthermore, our agreement algorithm works in anonymous networks where nodes do not know their neighbors and is also scalable, i.e., sends only a constant number of bits over an edge per round. We also show a matching lower bound of the agreement problem, which is optimal up to a $\mathrm{polylog}\, n$ factor.

The implicit leader election was studied in the *fault-free* setting (i.e., a network with no faulty nodes) by Kutten et al. [97]. They presented an optimal sublinear randomized algorithm and a tight lower bound on the message complexity (up to a $\mathrm{polylog}\, n$ factor). To the best of our knowledge, this is the first work that studies the message complexity of (implicit) leader election in the crash-fault synchronous distributed networks. We further show a non-trivial lower bound on the message complexity. The message complexity is optimal for more than a constant fraction of faulty nodes in the network (up to a $\mathrm{polylog}\, n$ factor). The $O(\log n)$ round complexity of both the problem (for any constant fraction of faulty nodes) is also almost optimal due to the lower bound $\Omega(\log n / \log \log n)$

by [37]. Our algorithms work in anonymous networks, where each node (initially) does not know the identity of its neighbors. Our algorithms are simple, lightweight (based on sampling only), and scalable (sends only $O(\log n)$ bits through an edge per round).

2.1.1 Our Results and Implications

We present sublinear algorithms and lower bounds on the message complexity of implicit agreement and leader election in a fully-connected synchronous network. Recall that the network has at least αn non-faulty nodes among n nodes[2] and the CONGEST communication model. The algorithms tolerate up to $(n - \log^2 n)$ faulty nodes. The algorithms and the lower bounds depend on the value of α, which essentially explains that the message complexity increases with the number of faulty nodes in the network. The main results are:

1. **Fault-tolerant leader election:** We present a randomized implicit leader algorithm which elects a leader with high probability in $O(\log n/\alpha)$ rounds and incurs $O\left((n^{1/2}\log^{5/2} n)/\alpha^{5/2}\right)$ messages, such that the elected leader is non-faulty with probability at least α. The message complexity is sublinear for $\alpha > (\log n/n^{1/5})$ and then the algorithm tolerates at most $(n - n^{4/5}\log n)$ faulty nodes. For any constant fraction faulty nodes in the network (i.e., when α is any constant), the algorithm is both message and time optimal (up to a $\mathrm{polylog}\, n$ factor). The algorithm can be extended to achieve explicit leader election in $O(\log n/\alpha)$ rounds and $O(n \log n/\alpha)$ messages.

2. **Message lower bound for fault-tolerant leader election:** We show an unconditional lower bound of $\Omega(n^{1/2}/\alpha^{3/2})$ messages for any leader election algorithm that succeeds with high probability and tolerates at most $(1 - \alpha)$-fraction faulty nodes. The lower bound also holds in the LOCAL model of distributed computing.

3. **Fault-tolerant agreement:** We present a randomized algorithm that solves (implicit) agreement with high probability in $O(\log n/\alpha)$ rounds and incurs $O\left((n^{1/2}\log^{3/2} n)/\alpha^{3/2}\right)$ messages with high probability. The algorithm is message optimal (up to a $\mathrm{polylog}\, n$ factor) as we show a tight lower bound. The message complexity is sublinear for $\alpha > \log n/n^{1/3}$, which means, the algorithm tolerates at most $(n - n^{2/3}\log n)$ faulty nodes and achieves sublinear message bound. The algorithm can be extended to achieve explicit agreement in $O(\log n/\alpha)$ rounds and $O(n \log n/\alpha)$ messages.

[2] We often use the notation αn to mean the integer number $\lceil \alpha n \rceil$ and $(1 - \alpha)n$ to mean the integer number $\lfloor (1 - \alpha)n \rfloor$.

4. **Message lower bound for fault-tolerant agreement:** We show an unconditional lower bound of $\Omega(n^{1/2}/\alpha^{3/2})$ on the message complexity for any agreement algorithm that succeeds with high probability and tolerates at most $(1-\alpha)$-fraction faulty nodes. The lower bound is essentially tight (up to a $\operatorname{polylog} n$ factor). The lower bound also holds in the LOCAL model.

One surprising fact about our results is that for any constant fraction of faulty nodes, the $\tilde{O}(n^{1/2})$ message complexity ($\tilde{O}$ hides a $\operatorname{polylog} n$ factor) of leader election and agreement is asymptotically same as in the fault-free network [97, 14]. The lower bounds are also matched.

Chapter Organization. Section 2.2 presents the model and definitions. Section 2.3 encloses the prior work in the direction. Section 2.4 contains fault-tolerant leader election algorithm (Section 2.4.1) and lower bound on the message complexity (Section 2.4.2). Section 2.5 contains fault-tolerant agreement algorithm (Section 2.5.1) and lower bound on the message complexity of agreement (Section 2.5.2). Finally, we conclude in Section 2.6.

2.2 Model and Definitions

The distributed network is fully connected (i.e., complete) n nodes graph. The network is anonymous, i.e., a node does not know the identity of its neighbors initially. This model is known as KT_0 (knowledge till 0 hop); on the other hand, in the KT_1 (knowledge till 1 hop) model, nodes know their neighbors [118]. A network is called f-resilient if at most f nodes may fail by crashing. We assume, up to $f \leq (n - \log^2 n)$ nodes may fail by crashing. More precisely, at most $(1-\alpha)n$ nodes can be faulty and the remaining at least αn nodes are non-faulty in the network, where α is a real constant that lies in the range $[\log^2 n/n,\ 1]$. The algorithm runs in synchronous rounds, and nodes communicate by exchanging messages in rounds. A faulty node may crash in any round during the execution of the algorithm. If a node crashes in a round, then an arbitrary subset (possibly all) of its messages for that round may be lost (as determined by an adversary) and may not be reached the destination. The crashed node halts (inactive) in the further rounds of the algorithm. If a node does not crash in a round, then all the messages that it sends in that round are delivered to the destination [11]. We assume a *static* adversary controls the faulty nodes, which selects the faulty nodes before the execution starts[3]. However, the adversary can adaptively choose when and how a node crashes. Each node has access to an arbitrary number of private random bits.

[3]In contrast, an adaptive adversary can determine the nodes to be faulty during the execution of the algorithm.

We consider *CONGEST* communication model [118], where a node is allowed to send a message of size $\mathcal{O}(\log n)$ bits through an edge per round. The message complexity of an algorithm is the total number of messages sent by all the nodes throughout the execution of the algorithm. Our lower bounds hold in the *LOCAL* communication model, where nodes are allowed to send a message of arbitrary size through an edge per round. We assume that nodes know the value of α and n. Although a node knows the value of n simply by its number of ports connected to the other nodes since the network is complete.

Let us now formally define the fault-tolerant leader election and agreement problem.

Definition 2.1 (Fault-Tolerant Leader Election)**.** *Consider a complete network of n nodes, in which at most f nodes could be faulty and the remaining $(n-f)$ nodes are non-faulty. The fault-tolerant leader election requires electing a leader node from the set of n nodes such that the leader is non-faulty with probability $(1-f/n)$* [4]*. Every node maintains a* state *variable that can be set to a value in $\{\perp, NONELECTED, ELECTED\}$, where $\perp$ denotes the 'undecided' state; initially, the state is set to $\perp$. Solving the fault-tolerant leader election problem requires the state of exactly one node to be $ELECTED$ and all other nodes are $NONELECTED$. This is the requirement for implicit leader election.*

Definition 2.2 (Fault-Tolerant Agreement)**.** *Consider a complete network of n nodes, in which at most f nodes could be faulty and the remaining $(n-f)$ nodes are non-faulty. Suppose initially each node has an input value in $\{0,1\}$. An implicit agreement holds when the final state is either $\{0,\perp\}$ or $\{1,\perp\}$ and at least one node has a state other than $\perp$ (which should be the input value of some node), where $\perp$ denotes the 'undecided' state. In other words, all the decided nodes must agree on the same value which is an initial input value of some node and there must be at least one decided node in the network.*

2.3 Related Work

Leader election has been extensively studied in various models and settings, starting from its introduction by Le Lann [101] in ring networks and then by the seminal work of Gallager-Humblet-Spira [54] in general graphs. The problem is well studied in the complete network itself [4, 69, 83, 85, 86, 130], and reference therein. Similarly, the agreement is also extensively studied in the class of complete networks [116, 106, 59, 37, 32, 33, 34, 36].

[4] Note that all the faulty nodes may survive and execute the algorithm correctly until the leader is elected, and then they may crash. Thus, a leader could be faulty with probability f/n if the adversary selects the faulty nodes uniformly at random in the beginning.

In case of agreement, Bar-Joseph and Ben-Or [17] showed the upper and lower bound $\Theta(f/\sqrt{n \log n})$ on the expected number of rounds in adaptive crash-fault. In the case of non-adaptive (i.e., static) adversary, Kowalski and Mirek [88] showed the upper and lower bound $\Theta(\sqrt{\frac{n}{(n-f)\log(n/(n-f))}})$ on the expected number of rounds. While these works are devoted to the round complexity, the main focus of this chapter is on message complexity. Recently, Hajiaghayi et al. [66] presents a crash-fault consensus algorithm having message complexity $\tilde{O}(n^{3/2})$ in a network where nodes know their neighbors' ID and the port connecting to it (a.k.a. KT_1 model) under adaptive adversary.

Let us discuss some relevant works that are closely related to ours. In a fault-free setting, Kutten et al. [97] studied the (implicit) leader election problem in complete networks and showed sub-linear message bounds using randomization, which breaks the linear lower bound of any deterministic algorithms (if we restrict on $O(1)$ time deterministic algorithm, it requires $\Omega(n^2)$ messages in a complete network). In particular, they presented an algorithm that executes in $O(1)$ time and uses only $O(n^{1/2}\log^{3/2} n)$ messages to elect a leader in a complete network where all the nodes are non-faulty. They also showed an almost matching lower bound for randomized leader election. Later, Augustine et al. [14] introduced and studied implicit agreement problem in fault-free settings and showed similar sub-linear bounds for agreement. Our work is inspired by the works of [97, 14, 59], but in the faulty setting.

In the crash-fault setting, our work is closely related to the work of Gilbert-Kowalski [59] which studied the message complexity of the agreement. They presented an optimal $\mathcal{O}(n)$ message complexity (explicit) agreement protocol which tolerates at most $n/2 - 1$ faulty nodes in the KT_1 model. The time complexity is $\mathcal{O}(\log n)$. While their algorithm solves the implicit agreement in a smaller set of nodes (of size $\mathcal{O}(\log n)$) in the course of the explicit agreement, they did not mention the implicit version of the agreement. Their implicit agreement solution uses $\tilde{\mathcal{O}}(n^{1/2})$ messages ($\tilde{\mathcal{O}}$ hides a $\operatorname{polylog} n$ factor) and $\mathcal{O}(\log n)$ rounds. In comparison, our agreement algorithm matches all these bounds while tolerating more faulty nodes.

The work [34] presented an (explicit) agreement algorithm tolerating a linear fraction of faulty nodes. The message complexity and the time complexity of their algorithm are $\mathcal{O}(n \log n)$ and $\mathcal{O}(\log n)$ respectively but in expectation. Recall that our result achieves the same bound with high probability and with higher resilience. The work [6] presented an agreement algorithm tolerating an arbitrary number of faults in the asynchronous and adaptive crash fault model, but their algorithm uses at least a quadratic number of messages (in expectation) when the number of faulty nodes is linear or more. Our agreement algorithm also gives better bounds compared to the deterministic algorithms presented in [33, 36, 35], in which the round complexity is $O(f)$ and the message complexity is $\tilde{\Omega}(n)$. Table 2.1 presents state-of-the-art in the same setting.

Comparative Analysis					
Paper	Message Complexity (in bits)	Round Complexity	Number of Faulty Nodes	Agreement Type	Model Type
Chlebus et. al, PODC'09 [36] #	$O(n \log^4 n)$	$O(f)$	$f < n$	Explicit	KT_1 [$]
Gilbert-Kowalski, SODA'10 [59]	$O(n)$	$O(\log n)$	$f < n/2$	Explicit	KT_1
Chlebus-Kowalski, SPAA'09 [34]	$O(n \log n)$*	$O(\log n)$*	$f < n/c$	Explicit	KT_0
Our Work	$O(n \log n)$	$O(\log n)$	$f < (1 - 1/c)n$	Explicit	KT_0
Our Work	$O(n \log n/\alpha)$	$O(\log n/\alpha)$	$f < (1 - \alpha)n$	Explicit	KT_0
Our Work	$O((n^{1/2} \log^{3/2} n)/\alpha^{3/2})$	$O(\log n/\alpha)$	$f < (1 - \alpha)n$	Implicit	KT_0

Table 2.1: Comparison with the best-known agreement protocols in the same model. Here, c is any constant and $\frac{\log^2 n}{n} \leq \alpha \leq 1$. # The algorithm is deterministic. * The bound holds in expectation. $ A recent brief-announcement paper [35] improved the message complexity to $O(n + f \log f)$, but missing formal proofs.

The leader election is a less studied problem in the faulty setting. In the *fault-free* setting, Gilbert et al. [60] provided an algorithm that solves implicit leader election in a general network with $O(\sqrt{n \log^7 n} \cdot t_{mix})$ messages and in $O(t_{mix} \log^2 n)$ rounds, where t_{mix} is the mixing time of the graph. Kowalski and Mosteiro [90] elect a unique leader using $\tilde{O}(\sqrt{n \cdot t_{mix}/\Phi})$ messages with high probability in the congest model, where Φ is the graph conductance.

We study the message complexity of the fault-tolerant leader election problem and show a non-trivial bound. The message complexity is optimal for any constant fraction of faulty nodes in the network (up to a $\text{polylog}\, n$ factor). The round complexity is almost optimal due to the lower bound $\Omega(\log n/\log\log n)$ [37].

In the context of Byzantine failure (where faulty nodes can behave maliciously), many excellent

works that studied the message complexity of leader election and agreement, a few of them are [1, 27, 72, 75, 76, 77, 78, 79].

2.4 Fault-Tolerant Leader Election

In this section, we first present a randomized algorithm that elects a leader with high probability in a complete n-node network with at least αn non-faulty nodes for a real value of α in $[\log^2 n/n, 1]$. Our goal is to minimize the message complexity of the algorithm while keeping the run time as small as possible. Then we show a non-trivial lower bound on the message complexity of any leader election algorithm.

2.4.1 Algorithm

The challenging part of designing an efficient algorithm is handling the faulty nodes. A faulty node may crash in some rounds and its message may not reach all the destination nodes in that round. Thus, there might be two sets of nodes with different states or views in the network. This may lead to an incorrect leader election. Let us describe the algorithm below.

Recall that we consider anonymous networks, i.e., nodes do not know each other (or, nodes do not have IDs), but know the values n and α. Initially, each node randomly picks an integer number in the range $[1, n^4]$ [5] (called its *rank*, which is also used as the ID of the node). The idea is to select the smaller committee nodes (called *candidate nodes*) which are responsible for electing a leader node among themselves. Among the candidate nodes, the node which proposes itself as a leader without crashing becomes the leader. A random set of candidate nodes of size $\Theta(\log n/\alpha)$ is selected, say, the set is $\mathcal{C}$. For this, each node selects itself with probability $O(\log n/\alpha n)$ to become a candidate node. The reason behind selecting so many candidate nodes is to make sure that $\mathcal{C}$ contains at least one non-faulty node. This selection is done locally at each node, and hence the candidate nodes do not know each other initially. Since knowing each other is message expensive, the candidate nodes communicate among themselves via some other nodes. For this, each candidate node randomly samples $\Theta(((n \log n)/\alpha)^{1/2})$ nodes among all the n nodes; call them as *referee nodes* and denoted by $\mathbb{R}$. The reason behind sampling so many referee nodes is to make sure at least one common "non-faulty" referee node between any pair of candidate nodes' referees. The candidate nodes communicate with each other via the referee nodes. Notice that a

[5] The range is taken in such a way that all the n ranks are distinct w.h.p. — which can be shown easily using a standard Chernoff bound [112].

node may be sampled as a referee node by multiple candidate nodes.

The idea is to make the minimum ID node among the candidate nodes to be the leader. That is why all the candidate nodes send their rank (which is also their ID) to the other candidate nodes. However, the minimum ID node may crash during the broadcast and its ID may not reach all the candidate nodes. Then the algorithm tries to select the second minimum node to be the leader, but it too may crash, and its ID may not be available to all the candidate nodes. Then the third minimum node and so on. This continues until it guarantees a node ID is available to all the candidate nodes so that they can agree on that node as the leader. Note that our algorithm promises that a crashed node is never elected as a leader (but it may crash after the election).

Pre-processing: First, each candidate node u sends its own rank, i.e., ID_u to its referee nodes, say $\mathbb{R}_u$. A referee node may receive ranks from multiple candidate nodes. Each referee node w forwards the list of received ranks to its respective candidate nodes, say $\mathcal{C}_w$. Thus, each candidate node has the list of some other candidate nodes' ranks; call the list as $rankList$[6]. While a candidate node knows the rank of another candidate node, this knowledge does not help to know the port (or the edge) connecting to that node. Now, the candidate nodes communicate via the referee nodes to agree on a leader from the $rankList$. This is done by performing the following steps iteratively.

Step 1: Each candidate node u proposes the minimum rank from its $rankList$ as a potential leader. The proposed rank of u, say, p_u is sent to all the candidate nodes via the referee nodes. For this, each candidate node u sends $\langle ID_u, p_u \rangle$ to its referee nodes $\mathbb{R}_u$. For a node u, if $p_u = ID_u$ i.e., u's rank is the minimum in its $rankList$, then u marks itself as the leader. During the iteration, if a candidate node receives a higher rank, it updates its $rankList$ by removing all the ranks smaller than the received rank. A node proposes a rank from its $rankList$ only once and does not propose the same rank in the later rounds.

Step 2: Then each referee node w sends the maximum proposed rank among the ranks that it received from its candidate nodes $\mathcal{C}_w$. Let $p_w^{\max}$ denote the maximum received rank by a referee node w. The referee node also sends the ID of a node u if u proposed its own rank (as the maximum rank $p_w^{\max}$), otherwise, sends a null value. More precisely, each referee node sends $\langle ID_u, p_w^{\max} \rangle$ if $p_w^{\max}$ is proposed by ID_u and $p_w^{\max} = ID_u$, otherwise sends $\langle \perp, p_w^{\max} \rangle$, where $\perp$ denotes a null value.

Step 3: Each candidate node u considers the maximum received rank, say, $\tilde{p}_u^{\max}$ from its referee nodes, i.e., $\tilde{p}_u^{\max} = \max\{p_w^{\max} : w \in \mathbb{R}_u\}$. If $ID_u = \tilde{p}_u^{\max}$ and u was not marked as the leader, then u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ to its referee nodes $\mathbb{R}_u$ in the next round and mark itself as the leader. If

[6] A candidate may not receive all other candidate nodes' rank due to faulty candidate nodes in $\mathcal{C}$.

$ID_u = \tilde{p}_u^{\max}$ and u was marked as the leader, then u does not respond in the next round. Otherwise, if $ID_u \neq \tilde{p}_u^{\max}$ then u decides the following. If u receives $\langle ID_v, \tilde{p}_u^{\max} \rangle$ from its referee nodes, i.e., $\tilde{p}_u^{\max}$ is proposed by some other candidate node v as its own proposed rank, then u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ and considers v as the leader until any further updates. The node v might crash in Step 1. In that case, any non-faulty nodes which do not receive v's rank may propose some higher rank in the next iteration. Thus, u might receive some higher rank later and accordingly updates its $rankList$ (i.e., removes the smaller ranks from the $rankList_u$). If u receives $\langle \perp, \tilde{p}_u^{\max} \rangle$ and $\tilde{p}_u^{\max} \in rankList_u$ then u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ in the next round, otherwise (if $\tilde{p}_u^{\max} \notin rankList_u$), u proposes a higher rank than $\tilde{p}_u^{\max}$ from its $rankList_u$ in the next round. Whenever u sends or proposes a higher rank, it updates its $rankList_u$ by removing the smaller ranks from it. If a node proposes itself as the potential leader successfully (i.e., without crashing) then the other nodes also agree on that node as the leader.

Step 4: If a candidate node u does not mark itself as the leader and does not propose another node as the leader in Step 1 and Step 3, and did not receive any updates in the next 4 rounds, then u sends the next minimum rank from its $rankList_u$ to $\mathbb{R}_u$. This case may arise when u proposed a minimum rank but does not receive the proposal from that particular node, or a node with a higher rank in the next 4 rounds. Then u will propose the next minimum rank available in its $rankList$. Since it might be the case that the rank which u proposed has crashed, which u can confirm after 4 rounds.

The above four steps are performed for $O(\log n/\alpha)$ iterations and the algorithm terminates. In the end, all the non-faulty candidate nodes have the same minimum rank in their $rankList$, which they agree to be the leader. Intuitively, there are $O(\log n/\alpha)$ candidate nodes and a single node may crash in each iteration (say, the minimum ID node crashes among the remaining candidate nodes). Since the set of candidate nodes contains at least one non-faulty node, the algorithm computes a unique leader after $O(\log n/\alpha)$ rounds (each iteration takes at most 4 rounds). Notice that if a candidate node u has not crashed in Step 1 and proposes its ID as the minimum rank, then u is going to be the leader since it successfully sends its rank to all the candidate nodes. It might crash in the next round, i.e., in Step 2 or later, but then all the non-faulty nodes are unaware of u's crash and u still be the unique leader in the network. A complete pseudocode is provided in the Algorithm 1,

Let us now show the few important lemmas that support the correctness of the algorithm. In particular, Lemma 2.2 shows that the set of candidate nodes contains at least one non-faulty node. Then lemma 2.3 shows that between any pair of candidate nodes, there is a common non-faulty referee node. It ensures that a non-faulty candidate node can successfully send messages to other

Algorithm 1 FAULT-TOLERANT LEADER ELECTION

Require: A complete n node anonymous network where at least $\lceil \alpha n \rceil$ nodes are non-faulty, where α is a real number in the range $[\log^2 n/n, 1]$. α is known to the nodes.
Ensure: Leader election.

1: Each node selects a rank uniformly at random from the range $[1, n^4]$. The rank becomes ID of the node.
2: Each node selects itself as a candidate node ($\mathcal{C}$) with probability $6 \log n/(\alpha n)$.
3: Each candidate node u randomly samples $2((n \log n)/\alpha)^{1/2}$ neighbors (called the referee nodes of u, $\mathbb{R}_u$) and sends its rank to the referee nodes.
4: Each referee node w sends all the received ranks to their respective candidate nodes, say $\mathcal{C}_w$, one by one. This takes at most $12 \log n/\alpha$ rounds.
5: Each candidate node maintains a list of ranks that received from its referee nodes. Each candidate node updates its $rankList$ by removing all the ranks smaller than the proposed rank. Potential leader suggested by any node is known as proposed rank.
6: Each candidate node u sends $\langle ID_u, p_u \rangle$ to its referees $\mathbb{R}_u$, where ID_u is the rank of u and p_u is the proposed rank which is the minimum rank in the current $rankList$ of u and update the $rankList$ of u (say $rankList_u$). ID_v is the rank of a specific candidate node v.
7: Each referee node w maintains $p_w^{\max} = \max\{p_u : u \in \mathcal{C}_w\}$. Also, $S^w \subseteq \mathcal{C}_w$ be the set of nodes which proposed the rank $p_w^{\max}$.
8: Each candidate node u maintains $\tilde{p}_u^{\max} = \max\{p_w^{\max} : w \in \mathbb{R}_u\}$. Also, $S^u \subseteq \mathbb{R}_u$ be the set of nodes which proposed the rank $\tilde{p}_u^{\max}$.
9: **for** the next $12 \log n/\alpha$ iteration, all nodes in parallel **do**
10: Each referee node w checks: ▷ nodes are active in every even round.
11: **if** $p_w^{\max} = ID_u \in S^w$ **then**
12: w sends $\langle ID_u, p_w^{\max} \rangle$ to $\mathcal{C}^w$.
13: **else** ▷ if there is no proposer in S^w which proposed the rank $p_w^{\max}$.
14: w sends $\langle \perp, p_w^{\max} \rangle$ to $\mathcal{C}_w$. ▷ $\perp$ denotes NULL ID.
15: **end if**
16: Each candidate node u checks: ▷ nodes are active in every odd round.
17: **if** $p_u = ID_u$ **then** ▷ u proposed itself as leader.
18: u send $\langle ID_u, p_u \rangle$ to $\mathbb{R}_u$ and become the leader.
19: **else if** $ID_u = \tilde{p}_u^{\max}$ and u is already leader **then**
20: u does not respond.
21: **else if** $ID_u = \tilde{p}_u^{\max}$ and u is not the leader **then**
22: u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ to $\mathbb{R}_u$ and become the leader.
23: **else if** $\tilde{p}_u^{\max} = ID_v \in S^u$ **then** ▷ some $\mathcal{C}$ proposed itself as leader.
24: u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ to $\mathbb{R}_u$ assuming v as leader till further update and update the $rankList$.
25: **else if** $\tilde{p}_u^{\max} \in rankList$ of u **then**
26: u sends $\langle ID_u, \tilde{p}_u^{\max} \rangle$ to $\mathbb{R}_u$ and update the $rankList$.
27: **else if** $\tilde{p}_u^{\max} \notin rankList$ of u **then**
28: p_u s.t. $p_u > \tilde{p}_u^{\max}$ and sends $\langle ID_u, p_u \rangle$ to $\mathbb{R}_u$ and update the $rankList$.
29: **end if**
30: **if** u does not know leader and did not get any update till next 4 rounds **then**
31: u propose the next minimum ID along with its ID to $\mathbb{R}_u$ and update the $rankList$.
32: **end if**
33: **end for**
34: All the non-faulty candidate nodes have the same minimum rank in their $rankList$ which they elect as leader.

candidate nodes via the common non-faulty referee node.

Lemma 2.1. *Consider an n-node network with at least αn non-faulty nodes, where α is a real number in the range $[\log^2 n/n, 1]$. If each node selects itself with probability $6\log n/(\alpha n)$ to become a candidate node then with high probability, i.e., with probability $\geq (1-1/n)$, the number of selected candidate nodes is $|\mathcal{C}| = \Theta(\log n/\alpha)$.*

Proof. Let a random variable X denotes the number of selected candidate nodes. Since each node selects itself independently with probability $6\log n/\alpha n$, the expected value of X is $E[X] = 6\log n/\alpha$. Thus, by using the Chernoff bound [112], $\Pr[X \geq (1+\delta)E[X]] \leq e^{-\delta^2 E[x]/3}$ for $\delta = 1$ we get,

$$\Pr[X \geq 12\log n/\alpha] \leq e^{-6\log n/3\alpha} < e^{-2\log n/\alpha} = \frac{1}{n^{2/\alpha}}.$$

Again using the Chernoff bound, $\Pr[X \leq (1-\delta)E[X]] \leq e^{-\delta^2 E[x]/2}$ for $\delta = 2/3$ we get,

$$\Pr[X \leq 2\log n/\alpha] \leq e^{-24\log n/18\alpha} < e^{-\log n/\alpha} = \frac{1}{n^{1/\alpha}}.$$

Thus, the size of the candidate set is $2\log n/\alpha \leqslant |\mathcal{C}| \leqslant 12\log n/\alpha$ with high probability for any value of $\alpha \leq 1$. □

Lemma 2.2. *The set of candidate nodes $\mathcal{C}$ contains at least one non-faulty node with high probability.*

Proof. Since $|\mathcal{C}|$ are selected independently and uniformly from n nodes and there are at least αn non-faulty nodes, the probability that all the nodes in $\mathcal{C}$ are faulty is at most $(1-(\alpha n)/n)^{|\mathcal{C}|}$, which is $e^{-|\mathcal{C}|\alpha} \leq e^{-2\log n} = 1/n^2$. That is, the probability that $\mathcal{C}$ has at least one non-faulty node is at least $1 - 1/n^2$. Thus, with high probability $\mathcal{C}$ contains at least one non-faulty node. □

Lemma 2.3. *Any pair of candidate nodes have at least one common non-faulty referee node with high probability.*

Proof. First, it follows from Lemma 2.2 that any $\mathbb{R}_u$ contains at least one non-faulty node, since $|\mathbb{R}_u| > |\mathcal{C}|$. Let us consider two candidate nodes, namely u and v. We show that there is at least one common non-faulty referee node for u and v, i.e., $\mathbb{R}_u \cap \mathbb{R}_v$ has at least one non-faulty node with high probability. Recall that each candidate node samples $2(n\log n/\alpha)^{1/2}$ referee nodes independently and uniformly among n nodes. As there are at least $\lceil \alpha n \rceil$ non-faulty nodes, the probability of sampling any non-faulty node as referee node is: $\frac{\alpha n}{n} . \frac{\left(2(n\log n/\alpha)^{1/2}\right)}{n}$. Thus, the probability that the

candidate node u is not selecting a non-faulty node as the referee node is $(1 - 2(\alpha \log n/n)^{1/2})$. It also holds for v.

Now the candidate node v samples $2(n \log n/\alpha)^{1/2}$ referee nodes. So, the probability of not selecting a non-faulty referee node from the sampled nodes of u is:

$$\left(1 - \frac{2(n \log n/\alpha)^{1/2}}{n/\alpha}\right)^{2(n \log n/\alpha)^{1/2}} \leq e^{-4 \log n} = \frac{1}{n^4}$$

Therefore, the probability of selecting at least one non-faulty referee node from the sampled nodes of u is at least $1 - 1/n^4$. Taking the union bound over all the pairs, the claim holds for any pair of candidate nodes. □

Finally, we show the round and message complexity of our algorithm in the main result of this section.

Theorem 2.1. *Consider a complete network of n nodes with at least $\lceil \alpha n \rceil$ non-faulty nodes and CONGEST communication model. Then there is a fault-tolerant leader election algorithm which elects a leader in $O(\log n/\alpha)$ rounds and incurs $O((n^{1/2} \log^{5/2} n)/\alpha^{5/2})$ messages with high probability, such that the elected leader is non-faulty with probability at least α, where α is real a number in $[\log^2 n/n, 1]$.*

Proof. The algorithm runs for $O(\log n/\alpha)$ iterations. Each iteration takes at most 4 rounds. There are some pre-processing steps before the iterations, which take $O(\log n/\alpha)$ rounds (since the referee nodes may need to send $O(\log n/\alpha)$ ranks to their candidate nodes in parallel). So the time complexity of our algorithm is $O(\log n/\alpha)$ rounds.

The size of the candidate nodes is $O(\log n/\alpha)$. Each candidate node samples $O(((n \log n)/\alpha)^{1/2})$ referee nodes and sends rank to them. Each referee node sends $O(\log n/\alpha)$ ranks to their respective candidate nodes. So the message complexity is $O((n^{1/2} \log^{5/2} n)/\alpha^{5/2})$ before the iterations start. In the iteration, the candidate nodes and the referee nodes communicate with each other for $O(\log n/\alpha)$ rounds, incurring $O((n^{1/2} \log^{5/2} n)/\alpha^{5/2})$ messages. Therefore, the total message complexity of the algorithm is $O((n^{1/2} \log^{5/2} n)/\alpha^{5/2})$.

Finally, we show the correctness of the algorithm. First, we show that all the non-faulty candidate nodes agree on a leader, and then show that the leader is unique. Suppose, a non-faulty candidate node, say u, does not have any knowledge about the leader. Then u proposes the minimum rank from its $rankList_u$ as a potential leader (see, Step 1). If u receives a higher rank from any other candidate node, say w, then u updates its leader accordingly (see, Step 3). Therefore, all

the candidate nodes must agree on a leader when the algorithm terminates. Now we show that the leader is unique. If not, then assume that there exist two candidate nodes u and v such that u agrees on a leader ℓ_1 and v agrees on a leader ℓ_2 at the end of the algorithm. Without loss of generality, assume that $\ell_1 > \ell_2$. This means that the rank ℓ_1 proposed by u is not reached to v. This implies that u has crashed during the iteration—which contradicts that u agrees on a leader at the end of the algorithm. Therefore, a crashed node is never elected as a leader, but it may crash later. Since at least αn nodes are non-faulty in the network, the probability that the elected leader is non-faulty is at least α. □

For any constant value of α, our algorithm solves the fault-tolerant leader election problem in $O(\log n)$ rounds and $O(n^{1/2}\log^{5/2} n)$ messages. Thus, we get the following corollary immediately.

Corollary 2.2. *Consider a complete network of n nodes and CONGEST communication model. There is a leader election algorithm which elects a leader in $O(\log n)$ rounds and incurs $O(n^{1/2}\log^{5/2} n)$ messages with high probability when tolerating at most $n - \lceil n/c \rceil$ faulty nodes for any constant $c \geq 1$. The elected leader is non-faulty with a probability of at least $1/c$.*

Remark: 1. *The above message bounds are in terms of the number of messages. Since the size of a message can be at most $O(\log n)$ bits (recall, it is a CONGEST model), the message complexity may be increased by a $O(\log n)$ factor in terms of bits.*

2.4.2 Lower Bound on the Message Complexity

We show a lower bound on the number of messages required to solve fault-tolerant leader election. In particular, we show that any algorithm, solving leader election with probability at least $2/e + \epsilon$, for any constant $\epsilon > 0$ and tolerates $(1-\alpha)n$ faulty nodes requires sending $\Omega(n^{1/2}/\alpha^{3/2})$ messages.

Our model assumes that all nodes execute the same algorithm and have access to an unbiased private coin (through which they can generate random bits locally). We assume for now that nodes are anonymous, i.e., nodes do not have IDs. Later we show by a simple reduction that the lower bound still holds even if the nodes start with unique IDs, but are not known to other nodes. The lower bound is unconditional on the running time and holds even in the *LOCAL* model of distributed computing[7] (which means that it also holds for the *CONGEST* model).

A brief overview of the proof. The basic proof idea is adapted from the paper [97], where they showed a message lower bound of $\Omega(n^{1/2})$ by any leader election algorithm in a "non-faulty"

[7] In *LOCAL* model, a node can send a message of arbitrary size through an edge per round.

network. While this $\Omega(n^{1/2})$ bound also holds in a faulty network, we improve the lower bound by a factor of $\alpha^{3/2}$ when there are at most $(1-\alpha)n$ faulty nodes in the network. Thus, our lower bound proof contains more technical challenges. For the sake of completeness, we include some results and arguments from [97]. The high-level idea is: if an algorithm does not send enough messages, i.e., communication among the nodes is less, then the algorithm might be wrong by electing multiple leaders. Since there might be two or more groups (of nodes) without any communication among them, and each group has an equal chance to elect a leader (as we assume that the nodes are anonymous). More precisely, suppose there is an algorithm $\mathcal{A}$ which solves leader election in this faulty network with $o(n^{1/2}/\alpha^{3/2})$ messages. Then we show a contradiction. For this, we consider a communication graph, which is essentially formed by the communication pattern of nodes during a run of $\mathcal{A}$ (an edge between two nodes if they exchange messages). Then we show that there are at least two disjoint components in the communication graph and each component has an equal probability to elect a leader in it – which leads to a contradiction. Thus, any algorithm that solves the leader election problem in this faulty network requires sending of $\Omega(n^{1/2}/\alpha^{3/2})$ messages. In fact, we show the following result.

Theorem 2.3. *Consider any algorithm $\mathcal{A}$ that solves leader election and sends at most $f(n)$ messages (of arbitrary size) with high probability on a complete anonymous network of n nodes with at least $\lceil \alpha n \rceil$ non-faulty nodes. If $\mathcal{A}$ solves leader election with probability at least $2/e + \epsilon$, for any constant $\epsilon > 0$, then $f(n) \in \Omega(n^{1/2}/\alpha^{3/2})$.*

Notice that the message lower bound is inversely proportional to the fraction of non-faulty nodes. This is intuitive, since, when the number of non-faulty nodes decreases (i.e., faulty nodes increase), there is a need of spending more resources (here it is messages) to mitigate the influence of the faulty nodes, and hence the message complexity increases. The remainder of the section concentrates on the proof of Theorem 2.3.

Assume, by contradiction, that there exists an algorithm $\mathcal{A}$ which solves the fault-tolerant leader election problem with probability at least $2/e + \epsilon$ and sends only $f(n) \in o(n^{1/2}/\alpha^{3/2})$ messages. We show a contradiction. Consider a complete network where for every node, the edges are randomly connected to the ports. In other words, an adversary chooses the connections of a node's $(n-1)$ ports as a random permutation over $\{1, 2, \ldots, n-1\}$.

We adapt the following definitions from [97]. "Consider one (arbitrary but fixed) execution of the algorithm $\mathcal{A}$. Let us define the *communication graph* $\mathcal{C}^r$ to be a directed graph on the given set of n nodes, where there is an edge from u to v if and only if u sends a message to v in some round $r' \leq r$. For any node u, denote the state of u in round r by $\sigma_r(u)$. Let Σ be the set of all nodes' states possible in the algorithm $\mathcal{A}$. We say that node u *influence nodes* w by round r if there is a

directed path from u to w in $\mathcal{C}^r$. A node u is called an *initiator* if it is not influenced before sending its first message. That is, if u sends its first message in round r, then u has an outgoing edge in $\mathcal{C}^r$ and is an isolated vertex in $\mathcal{C}^1, \ldots, \mathcal{C}^{r-1}$. For every initiator u, we define the influence cloud $\mathcal{IC}_u^r$ as the pair $\mathcal{IC}_u^r = (\mathcal{C}_u^r, \mathcal{S}_u^r)$, where $\mathcal{C}_u^r = \langle u, w_1, \ldots, w_k \rangle$ is the ordered set of all nodes that are influenced by u, namely, that are reachable along a directed path in $\mathcal{C}^r$ from u ordered by the time by which they joined the cloud (breaking ties arbitrarily), and $\mathcal{S}_u^r = \langle \sigma_r(u), \sigma_r(w_1), \ldots, \sigma_r(w_k) \rangle$ is their configuration after round r, namely, their current tuple of states. We may sometimes abuse notation by referring to the ordered node set $\mathcal{C}_u^r$ as the influence cloud of u. Note that a passive (non-initiator) node v does not send any messages before receiving the first message from some other node."

There must be enough initiator nodes, otherwise, if all the initiator nodes crash, the algorithm may not succeed with high probability. In fact, we show that in a network with at least αn non-faulty nodes, any leader election algorithm which succeeds with at least a constant probability requires at least $1/2\alpha$ initiator nodes.

Lemma 2.4. *Any leader election algorithm in this crash-fault model requires at least* $\lceil 1/2\alpha \rceil$ *initiator nodes (*$\alpha < 1/24$*) to guarantee at least a constant success probability.*

Proof. It must be shown that there is at least one non-faulty initiator node; otherwise, if all the initiator nodes are faulty, then they may crash at the beginning of the algorithm (in the worst case). Without loss of generality, assume that the initiator nodes are chosen uniformly at random. Let E be the event that there is at least one non-faulty initiator node among the $\lceil 1/2\alpha \rceil$ nodes when chosen uniformly at random. Then,

$$\Pr[E] = 1 - \left(1 - \frac{\alpha n}{n}\right)^{\lceil 1/2\alpha \rceil} = 1 - \left(1 - \alpha\right)^{\lceil 1/2\alpha \rceil} \geq 1 - e^{-1/2}$$

Thus, the algorithm needs at least $\lceil 1/2\alpha \rceil$ initiator nodes so that at least one of them is non-faulty with probability at least $1 - e^{-1/2}$, a constant. Hence, the lemma. □

Since we assume that $\mathcal{A}$ sends $o(n^{1/2}/\alpha^{3/2})$ messages – a finite number of messages, there is some round ρ by which no more messages are sent. Let us assume $\mathcal{C}_{u_1}^\rho, \mathcal{C}_{u_2}^\rho, \ldots, \mathcal{C}_{u_i}^\rho$, $(i < 1/2\alpha)$ are at most $1/2\alpha$ influence clouds corresponding to the initiator nodes in the end of the algorithm. Recall that some influence clouds may merge during the execution. Therefore, the size of the smallest influence cloud, say, $\mathcal{C}_{u_*}^\rho$ is at most $o(\frac{n^{1/2}/\alpha^{3/2}}{1/2\alpha})$, which is $o(n^{1/2}/\alpha^{1/2})$.

In general, it is possible that in a given execution, two influence clouds $\mathcal{C}_{u_1}^r$ and $\mathcal{C}_{u_2}^r$ intersect

each other over some common node v, if v happens to be influenced by both u_1 and u_2. The following lemma shows that the low message complexity of the algorithm $\mathcal{A}$ yields a good probability for the smallest influence cloud $\mathcal{C}^{\rho}_{u_*}$ to be disjoint from all other influence clouds.

Let N be the event that there is no intersection between (the nodes of) all the influence clouds and the smallest influence cloud, i.e., $\left(\cup_{i:i\neq *}\mathcal{C}^{\rho}_{u_i}\right)\cap\mathcal{C}^{\rho}_{u_*}=\emptyset$. Let M be the event that the algorithm $\mathcal{A}$ sends no more than $f(n)$ messages.

Lemma 2.5. *Assume that* $\Pr[M]\geq\left(1-\frac{1}{n}\right)$. *If* $f(n)\in o\big(n^{1/2}/\alpha^{3/2}\big)$, *then* $\Pr[N\wedge M]\geq 1-\frac{1}{n}-\frac{12\alpha^3 f^2(n)}{n}+\frac{12\alpha^3 f^2(n)}{n^2}\in 1-o(1)$.

Proof. Consider a round r, the smallest influence cloud $\mathcal{C}^r$ at round r, and a node $v\in\mathcal{C}^r$. Without loss of generality assume that the smallest influence cloud is unique, otherwise, fix one arbitrarily. Assuming an event M, there are at most $f(n)$ nodes that have sent or received a message from any influence clouds and may thus be a part of some other cloud except $\mathcal{C}^r$. Recall that the port numbering of every node was chosen uniformly at random, and we conditioned it on the occurrence of the event M. The initiator node of the smallest influence cloud may know the destinations of at most $2\alpha f(n)$ of its ports in any round; out of them $2\alpha^2 f(n)$ ports are connected to non-faulty nodes in expectation. Then it is easy to show using Chernoff bound (see Theorem 4.4 in [112]) that at most $12\alpha^2 f(n)$ ports are connected to non-faulty nodes with high probability. Similarly, there are at least $\alpha f(n)/2$ non-faulty nodes among the $f(n)$ nodes with high probability (using the Chernoff bound in Theorem 4.5, [112]). Therefore, to send a message from the smallest influence cloud to another cloud, v must hit upon one of the ports which connected to a non-faulty node leading to other clouds. Let H be the event that a message sent by a node v in round r reaches a node u that is already part of some other (non-singleton) cloud. (Recall that if u is in a singleton cloud due to not having received or sent any messages yet, it simply becomes a member of v's cloud and the smallest influence cloud does not send a message to the singleton cloud of the initiator node, otherwise, it will not be the smallest influence cloud). Therefore, the number of non-faulty nodes in the influence clouds except the nodes in the smallest influence cloud is: $\frac{\alpha f(n)}{2}-12\alpha^2 f(n)$ which is $\frac{\alpha(1-24\alpha)f(n)}{2}$. In consequence, we have the probability that the smallest influence cloud has to collude with any of the $(1-2\alpha)/2\alpha$ (i.e., $1/2\alpha-1$) clouds:

$$\Pr\left[H|M\right]\leq 2\sum_{i=1}^{\frac{\alpha(1-24\alpha)f(n)}{2}}12\alpha^2 f(n)\cdot\frac{1}{n}\;\leq\frac{12\alpha^3 f^2(n)(1-24\alpha)}{n}<\frac{12\alpha^3 f^2(n)}{n}$$

which is $o(1)$ as $f(n)\in o(n^{1/2}/\alpha^{3/2})$.

In the above inequality, the probability $12\alpha^2 f(n)$ is the number of non-faulty nodes (w.h.p.) in the smallest influence cloud and $1/n$ is the probability of selecting a particular node by another good node. Further, we have $\frac{\alpha(1-24\alpha)f(n)}{2}$ non-faulty nodes w.h.p. which can collude with the smallest influence cloud throughout the execution of the algorithm $\mathcal{A}$. Moreover, a node in the smallest influence cloud might receive messages from other influence clouds for the collision and vice versa. Consequently, we have a factor 2 for both ways whether the message is received or sent by the cloud.

Observe that $\Pr[N|M] = 1 - \Pr[H|M]$. Since $\Pr[N \wedge M] = \Pr[N|M].P[M]$. It follows that $\Pr[N \wedge M] \geq (1 - \frac{12\alpha^3 f^2(n)}{n})(1 - \frac{1}{n}) \geq 1 - \frac{1}{n} - \frac{12\alpha^3 f^2(n)}{n} + \frac{12\alpha^3 f^2(n)}{n^2} \in 1 - o(1)$, as required. □

We adapt the following configuration from [97]. This adaptation is helpful to generate different configurations which help to show that there exists a cloud configuration that is disjoint from every other configuration, i.e., has no information about the other configuration's decision. "We next consider *potential cloud configurations*, namely, $Z = \langle \sigma_0, \sigma_1, \ldots, \sigma_{1/2\alpha} \rangle$, where $\sigma_i \in \Sigma$ or every i, and more generally, potential cloud configuration sequences $\bar{Z}^r = Z^1, \ldots, Z^r$, where each Z^i is a potential cloud configuration, which may potentially occur as the configuration tuple of some influence clouds in a round i of some execution of the algorithm $\mathcal{A}$. We study the occurrence probability of potential cloud configuration sequences.

We say that the potential cloud configuration $Z = \langle \sigma_0, \sigma_1, \ldots, \sigma_{1/2\alpha} \rangle$ is realized by the initiator u in round r if the influence cloud $\mathcal{IC}_u^r = (\mathcal{C}_u^r, \mathcal{S}_u^r)$ has the same node states in $\mathcal{S}_u^r$ as those of Z, or more formally $\mathcal{S}_u^r = \langle \sigma_r(u), \sigma_r(w_1), \ldots, \sigma_r(w_{1/2\alpha}) \rangle$ such that $\sigma_r(u) = \sigma_0$ and $\sigma_r(w_i) = \sigma_i$ for every $i \in [1, 1/2\alpha]$. In this case, the influence cloud $\mathcal{IC}_u^r$ is referred to as a realization of the potential cloud configuration Z. (Note that a potential cloud configuration may have many different realizations.)

More generally, we say that the potential cloud configuration sequence $\bar{Z}^r = (Z^1, \ldots, Z^r)$ is realized by the initiator u if for every round $i = 1, ..., r$, the influence cloud $\mathcal{IC}_u^i$ is a realization of the potential cloud configuration Z^i. In this case, the sequence of influence clouds of u up to round r, $\bar{\mathcal{IC}}_u^r = \langle \mathcal{IC}_u^1, \ldots, \mathcal{IC}_u^r \rangle$, is referred to as a realization of $\bar{Z}^r$. (Recall, a potential cloud configuration sequence may have many different realizations.)

For a potential cloud configuration Z, let $E_u^r(Z)$ be the event that Z is realized by the initiator u in (round r of) the run of algorithm $\mathcal{A}$. For a potential cloud configuration sequence Z^r, let $E_u(Z^r)$ denote the event that Z^r is realized by the initiator u in (the first r rounds of) the run of algorithm $\mathcal{A}$."

Lemma 2.6. *Every initiator node has the same probability to be part of the smallest influence*

cloud.

Proof. Let $a_1, a_2, \ldots, a_{1/2\alpha}$ are the initiator nodes. Let E_i and E_j be the event that the initiator node a_i and a_j are part of the smallest influence cloud where $i \neq j$. Recall, each node has the same probability to be an initiator node. Also, each initiator node has the same probability to be a non-faulty node. Suppose $\Pr[E_i] < \Pr[E_j]$ without loss of generality. This means that node a_j has a higher probability to be in the smallest influence cloud than node a_i. Thus, in this way, it is possible to estimate the size of the influence clouds in a biased way which essentially contradicts the uniformity of the initiator nodes. Therefore, $\Pr[E_i] = \Pr[E_j]$. □

Moreover, the following Lemma 2.7 and supported text is adapted from [97]. This adaptation helps us show that there exist equal probability for any initiator to be disjoint from other influence clouds in faulty setting.

Lemma 2.7. *"Restrict attention to executions of algorithm $\mathcal{A}$ that satisfy event N, namely, in which any influence cloud is disjoint from other influence clouds. Then $\Pr[E_u(\bar{Z}^r)] = \Pr[E_v(\bar{Z}^r)]$ for every $r \in [1, \rho]$, every potential cloud configuration sequence $\bar{Z}^r$, and every two initiators u and v.*

Proof. The proof is by induction on r. Initially, in round 1, all possible influence clouds of the algorithm $\mathcal{A}$ are singletons, i.e., their node sets contain just the initiator. Neither u nor v has received any messages from other nodes. This means that $\Pr[\sigma_1(u) = s] = \Pr[\sigma_1(v) = s]$ for all $s \in \Sigma$, thus any potential cloud configuration $Z^1 = \langle s \rangle$ has the same probability of occurring for any initiator, implying the claim.

Assuming that the result holds for round $r - 1 \geq 1$, we show that it still holds for round r. Consider a potential cloud configuration sequence $\bar{Z}^r = (Z^1, \ldots, Z^r)$ and two initiators u and v. We need to show that $\bar{Z}^r$ is equally likely to be realized by u and v, conditioned on the event N. By the inductive hypothesis, the prefix $\bar{Z}^{r-1} = (Z^1, \ldots, Z^{r-1})$ satisfies the claim. Hence, it suffices to prove the following. Let p_u be the probability of the event $E_u^r(Z^r)$ conditioned on the event $N \wedge E_u^r(\bar{Z}^{r-1})$. Define the probability p_v similarly for v. Then it remains to prove that $p_u = p_v$.

To do that we need to show, for any state $\sigma_j \in Z^r$, that the probability that $w_{u,j}$ the jth node in $\mathcal{IC}_u^r$ is in state σ_j, conditioned on the event $N \wedge E_u(\bar{Z}^{r-1})$, is the same as the probability that $w_{v,j}$, the jth node in $\mathcal{IC}_v^r$ is in state σ_j, conditioned on the event $N \wedge E_v(\bar{Z}^{r-1})$.

There are two cases to be considered. The first is that the potential influence cloud Z^{r-1} has j or more states. Then by our assumption that events $E_u(\bar{Z}^{r-1})$ and $E_v(\bar{Z}^{r-1})$ hold, the nodes $w_{u,j}$ and $w_{v,j}$ were already in u's and v's influence clouds, respectively, at the end of the round

$r-1$. The node $w_{u,j}$ changes its state from its previous state, σ'_j, to σ_j on round r as the result of receiving some messages $M_1, \ldots, M_\ell$ from neighbors $x_1^u, ..., x_\ell^u$ in u's influence cloud $\mathcal{IC}_u^{r-1}$, respectively. In turn, the node x_j^u sends a message M_j to $w_{u,j}$ on round r as the result of being in a certain state $\sigma_r(x_j^u)$ at the beginning of round r (or equivalently, at the end of round $r-1$) and making a certain random choice (with a certain probability q_j for sending M_j to $w_{u,j}$). But if one assumes that the event $E_v(\bar{Z}^{r-1})$ holds, namely, that $\bar{Z}^{r-1}$ is realized by the initiator v, then the respective nodes $x_1^v, \ldots, x_\ell^v$ in v's influence cloud $\mathcal{IC}_v^{r-1}$ will be in the same respective states ($\sigma_r(x_j^v) = \sigma_r(x_j^u)$ for every j) on the end of round $r-1$, and therefore will send the messages $M_1, \ldots, M_\ell$ to the node $w_{v,j}$ with the same probabilities q_j. Also, on the end of round $r-1$, the node $w_{v,j}$ is in the same state σ'_j as $w_{u,j}$ (assuming event $E_v(\bar{Z}^{r-1})$). It follows that the node $w_{v,j}$ changes its state to σ_j on round r with the same probability as the node $w_{u,j}$.

The second case to be considered is when the potential influence cloud Z^{r-1} has fewer than j states. This means (conditioned on the events $E_u(\bar{Z}^{r-1})$ and $E_v(\bar{Z}^{r-1})$ respectively) that the nodes $w_{u,j}$ and $w_{v,j}$ were not in the respective influence clouds on the end of the round $r-1$. Rather, they were both passive nodes. By an argument similar to that made for round 1, any pair of (so far) passive nodes have equal probability of being in any state. Hence, $\Pr[\sigma_{r-1}(w_{u,j}) = s] = \Pr[\sigma_{r-1}(w_{v,j}) = s]$ for all $s \in \Sigma$. As in the former case, the node $w_{u,j}$ changes its state from its previous state, σ'_j to σ_j on round r as the result of receiving some messages $M_1, \ldots, M_\ell$ from neighbors $x_1^u, \ldots, x_\ell^u$ that are already in u's influence cloud $\mathcal{IC}_u^{r-1}$, respectively. By a similar analysis, it follows that the node $w_{v,j}$ changes its state to σ_j on round r with the same probability as the node $w_{u,j}$. □

We now conclude that for any potential cloud configuration Z and any two initiators u and v, the events $E_u^\rho(Z)$ and $E_v^\rho(Z)$ are equally likely. More specifically, we say that the potential cloud configuration Z is equiv-probable for initiators u and v if $\Pr[E_u^\rho(Z)|N] = \Pr[E_v^\rho(Z)|N]$. Although a potential cloud configuration Z may be the end-cloud of many different potential cloud configuration sequences, and each such potential cloud configuration sequence may have many different realizations, the above lemma implies the following (integrating over all possible choices)."

Corollary 2.4. *Restrict attention to executions of algorithm $\mathcal{A}$ that satisfy event N, namely, in which the smallest influence cloud is disjoint from the other influence cloud. Consider two initiators u and v and a potential cloud configuration Z. Then Z is equiv-probable for u and v.*

By assumption, the algorithm $\mathcal{A}$ succeeds with probability at least $2/e + \epsilon$, for some fixed constant $\epsilon > 0$. Let S be the event that $\mathcal{A}$ elects exactly one leader. We have

$$\begin{aligned}\frac{2}{e}+\epsilon \leq \Pr[S] &= \Pr[S|M \wedge N]\Pr[M \wedge N] \\ &\quad + \Pr[S|not(M \wedge N)][not(M \wedge N)] \\ &\leq \Pr[S|M \wedge N]\Pr[M \wedge N] + [not(M \wedge N)].\end{aligned}$$

By Lemma 2.5, we know that $\Pr[M \wedge N] \in 1 - o(1)$ and $\Pr[not(M \wedge N)] \in o(1)$, and thus it follows that

$$\Pr[S|M \wedge N] \geq \frac{2/e + \epsilon - o(1)}{1 - o(1)} > \frac{2}{e}, \text{ for sufficiently large } n. \tag{2.1}$$

By Corollary 2.4, each of the initiators has the same probability p of realizing a potential cloud configuration where some node is a leader. Assuming that events M and N occur, it is immediate that $0 < p < 1$. Let X be the random variable that represents the smallest influence cloud is disjoint from the other influence clouds. Recall that the algorithm $\mathcal{A}$ succeeds whenever an event S occurs. Its success probability that the smallest influence cloud is disjoint from the other influence cloud, is given by,

$$\Pr[S|M \wedge N] = 2p(1-p) \tag{2.2}$$

The probability value in Equation 2.2 is maximum when $p = 1/2$, which yields that $\Pr[S|M \wedge N] \leq 1/2$. This, however, is a contradiction to Equation 2.1. Thus, our assumption that there is an algorithm $\mathcal{A}$ that solves leader election with probability at least $2/e + \epsilon$ using only $o(n^{1/2}/\alpha^{3/2})$ messages is wrong. That is, any algorithm incurs $\Omega(n^{1/2}/\alpha^{3/2})$ messages. This completes the proof of Theorem 2.3.

Now we argue using a standard technique that the above lower bound holds for any algorithm which assumes that nodes are equipped with unique IDs. For this, we simply assume that an adversary provides IDs, chosen uniformly at random from, say, $[1, n^4]$. Let $\mathbb{B}$ be an algorithm that exploits these unique IDs. Notice, however, that the execution of $\mathbb{B}$ that exploits the adversarially generated random IDs is essentially equivalent to the execution of $\mathbb{B}$ in the anonymous setting, in which each node first generates a random number between $[1, n^4]$ and then uses that random number as their IDs. The only difference is that multiple nodes may generate the same random number, thereby inheriting the same ID, but this is an event whose probability is at most $1/n$ (this can be easily shown using Chernoff bound as the range is taken $[1, n^4]$). So when we condition these random numbers being distinct, the two executions are probabilistically identical, so any

lower bound on the execution without adversarially generated IDs will extend to the case where nodes have unique IDs provided by the adversary. A more formal argument can be found in [97].

2.5 Fault-Tolerant Agreement

In this section, we first present a randomized algorithm that solves agreement with high probability in a complete n-node network with at most $(1-\alpha)n$ faulty nodes, where α is a real number in $[\log^2 n/n, 1]$. The algorithm takes $O(\log n/\alpha)$ rounds and sends no more than $O(n^{1/2}\log^{3/2} n/\alpha^{3/2})$ messages with high probability. Our algorithm is message optimal (up to a $\operatorname{polylog} n$ factor) as we also show a matching lower bound $\Omega(n^{1/2}/\alpha^{3/2})$ on the message complexity.

Note that a leader election algorithm immediately gives a solution to the agreement problem: simply by agreeing on the leader's input value. Hence, our leader election algorithm also solves agreement, but then the message complexity would be $O(n^{1/2}\log^{5/2} n/\alpha^{5/2})$. We design an efficient algorithm which solves agreement using $O(n^{1/2}\log^{3/2} n/\alpha^{3/2})$ messages in this faulty network. On the other hand, a lower bound of the leader election problem does not apply to the agreement as the agreement is an easier problem than the leader election. We show a lower bound $\Omega(n^{1/2}/\alpha^{3/2})$ on the message complexity of the fault-tolerant agreement problem.

2.5.1 Algorithm

Let us consider the binary agreement problem. Recall that we consider an anonymous network, i.e., nodes do not know each other but know the values of n and α. Initially, each node receives a value in $\{0,1\}$ given by an adversary. Our goal is to design a message efficient algorithm that solves agreement implicitly. At the end of the algorithm, a non-empty subset of the nodes must agree on a single value (between 0 and 1) while preserving the validity condition.

Let $b_u \in \{0,1\}$ denote the input bit of a node u. The basic framework of the algorithm is similar to the leader election algorithm in Section 2.4.1. First, select a smaller committee of nodes, called the *candidate nodes* which agree on a value among themselves. For this, a random set of candidate nodes of size $\Theta(\log n/\alpha)$ is selected, say the set is $\mathcal{C}$. Since the candidate nodes do not know each other, they communicate among themselves via referee nodes. For this, each candidate node samples $O(n^{1/2}\log^{1/2} n/\alpha^{1/2})$ *referee nodes* from its neighbors. The set of referee nodes of a candidate node u is denoted by $\mathbb{R}_u$. Recall that a node may be sampled as a referee node by multiple candidate nodes.

The idea of the algorithm is: the candidate nodes are biased to agree on 0 if any of them receives 0 as an input bit. For this, any candidate node whose input value is 0 forwards 0 to all the candidate nodes. If there is at least one candidate node with value 0, then 0 is propagated to all the candidate nodes eventually, and they agree on 0. If none of the candidate nodes receives 0 as an input (i.e., all of them get 1), they do not send any messages and agree on 1 eventually.

Step 0: Each candidate node u does in parallel: if $b_u = 0$ then u sends 0 to its referee nodes $\mathbb{R}_u$ and agrees on 0. If $b_u = 1$ then u sends 1 to $\mathbb{R}_u$, but does not agree on 1. This step is required to inform the nodes in $\mathbb{R}_u$ that they are selected as referee nodes by u.

Then perform the following steps iteratively.

Step 1: If a candidate node u receives 0 from any of its referee nodes, and it has not agreed on 0 before, then u sends 0 to its referee node and agrees on 0. If u has agreed on 0 before, then u does not send anything.

Step 2: If a referee node w possesses 0 from any of its candidate nodes, and it has not sent 0 earlier, then w sends 0 to its candidate nodes, say, $\mathcal{C}_w$.

The above two steps (Step 1 and 2) are performed for $O(\log n/\alpha)$ iterations, and then the algorithm terminates. In the end, all the (non-faulty) candidate nodes have the same minimum value, either 0 or 1. If they do not have 0, they agree on 1. Intuitively, there are $O(\log n/\alpha)$ candidate nodes and a single node may crash in each iteration (say, the single node with value 0 crashes among the remaining candidate nodes) and thus, 0 may propagate slowly. But eventually, 0 reaches to all the candidate nodes as the set of candidate nodes contains at least one non-faulty node (see Lemma 2.2) and there is a common non-faulty referee node between any pair of candidate nodes (see Lemma 2.3). On the other hand, if all the candidate nodes possess input value 1, then the algorithm does not send any messages during the iterations and terminates after $O(\log n/\alpha)$ rounds with all the candidate nodes agreeing on 1. Thus, the algorithm agrees on a unique value (which must be an input) after $O(\log n/\alpha)$ rounds (as each iteration takes at most 2 rounds). A complete pseudocode is provided in the Algorithm 2.

It is easy to see that the algorithm terminates in $O(\log n/\alpha)$ rounds. In Step 0, at most $O(\log n/\alpha)$ candidate nodes send a message to $O(((n \log n)/\alpha)^{1/2})$ referee nodes. Later in the iteration, a candidate node may send a message to its referee nodes at most once – when it receives a 0 from another candidate node (via the referee nodes). The same is true for a referee node. A referee node only sends the received value 0 to its candidate nodes only once. Also, note that all the messages contain only a single-bit value. Therefore, the total message complexity of the algorithm is $O(n^{1/2} \log^{3/2} n/\alpha^{3/2})$ bits. Thus, we get the following main result of this section.

Algorithm 2 FAULT-TOLERANT-AGREEMENT

Require: A complete n-node anonymous network with at least $\lceil \alpha n \rceil$ non-faulty nodes, α is known to the nodes. Each node receives a value in $\{0, 1\}$ given by an adversary.
Ensure: Implicit Agreement.

1: Each node selects itself as a candidate node ($\mathcal{C}$) with probability $6 \log n/(\alpha n)$.
2: Each candidate node u randomly samples $2((n \log n)/\alpha)^{1/2}$ neighbours (called referee nodes of u, denoted by $\mathbb{R}_u$).
3: Each candidate node u sends its input bit $b_u \in \{0, 1\}$ to its $\mathbb{R}_u$. If $b_u = 0$, then u agrees on 0.
4: **for** the next $12 \log n/\alpha$ iteration, all nodes in parallel **do**
5: Each referee node w checks:
6: **if** w possess the value 0 and have not send to $\mathcal{C}_w$ **then** ▷ $\mathcal{C}_w$ is the set of candidate nodes for referee node w.
7: w sends 0 to $\mathcal{C}_w$.
8: **else**
9: w does not send the value.
10: **end if**
11: Each candidate node u checks:
12: **if** u receives 0 before reaching at agreement **then**
13: u sends 0 to $\mathbb{R}_u$ and agree on the value 0.
14: **else if** u receives 0 after reaching at agreement **then**
15: u ignores 0.
16: **end if**
17: **end for**
18: All non-faulty candidate nodes have the same minimum value on which they agree.

Theorem 2.5. *Consider a complete n-node network with at least $\lceil \alpha n \rceil$ non-faulty nodes and CONGEST communication model. Then there is a randomized algorithm which solves agreement with high probability in $O(\log n/\alpha)$ rounds and incurs $O\left(n^{1/2} \log^{3/2} n/\alpha^{3/2}\right)$ message bits with high probability.*

For any constant value of α, the algorithm solves the fault-tolerant agreement in $O(\log n)$ rounds and $O(n^{1/2} \log^{3/2} n)$ messages. Thus, we get the following corollary immediately.

Corollary 2.6. *Consider a complete n-node network and CONGEST communication model. There is a randomized algorithm which solves agreement with high probability in $O(\log n)$ rounds and incurs $O(n^{1/2} \log^{3/2} n)$ message bits with high probability when tolerating at most $\lfloor n - n/c \rfloor$ faulty nodes for any constant $c \geq 1$.*

Implicit to Explicit Agreement. The above result accomplishes sub-linear message bound (when

$\alpha = o(\log n/n^{1/3})$) for the implicit agreement. For the explicit agreement, the message lower bound is $\Omega(n)$, since the agreed value needs to reach to all the nodes in the network. The above implicit agreement algorithm can be easily extended to an explicit agreement using $O(n \log n/\alpha)$ messages in one extra round. At the end of the implicit agreement algorithm, the candidate nodes which reach an agreement broadcast the agreed value to all the nodes in the network in parallel in a single round. Then all the nodes agree on the received value. Since the size of such candidate nodes is at most $O(\log n/\alpha)$, the message complexity of this step is $O(n \log n/\alpha)$. Hence, the claim.

2.5.2 Lower Bound on the Message Complexity

We show that any algorithm for implicit agreement with constant success probability requires $\Omega(n^{1/2}/\alpha^{3/2})$ messages to be sent in order to achieve implicit agreement with probability at least $1 - \epsilon$ for a sufficiently small $\epsilon > 0$. Section 2.4.2 shows that leader election requires $\Omega(n^{1/2}/\alpha^{3/2})$ messages, but our proof is different because, under leader election, all nodes essentially start in identical configurations. In the implicit agreement problem, however, the nodes start with initial values and our lower bound argument takes that into account and shows that no algorithm can exploit those initial values in order to reach agreement with fewer messages. Our approach and writing style are highly inspired/adapted by the work done in [14]. Informally, we are adapting the work of [14] to address the specific challenges posed by faulty setting.

We initiate the setup with the formation as discussed in [14] which can be adapted in faulty setting as well. "Consider a randomized algorithm $\mathcal{A}$ to achieve implicit agreement with probability $1 - \epsilon$. For the sake of contradiction, let us assume that $\mathcal{A}$ sends at most $o(n^{1/2}/\alpha^{3/2})$ messages with probability at least $1 - o(1)$. We will show that $\mathcal{A}$ must have an error probability that is at least a constant, so for the sake of contradiction, assume that the probability with which $\mathcal{A}$ does not reach implicit agreement is at most $o(1)$. For now, we will assume that the nodes are anonymous, so $\mathcal{A}$ does not have access to unique node identifiers or any other identifying feature pertaining to individual nodes.

Recall also that the network is anonymous, so a node u, at least at the start of the protocol, will not know which incident edge leads to some neighbor v. In fact, for the purpose of showing this lower bound, we assume that for every node, the neighbor sequence (as we enumerate from port 1 to port $n - 1$) is a uniformly random permutation independent of all other nodes' permutations.

Given any $p \in [0, 1]$, let C_p denote the (random) starting configuration wherein each node is independently assigned an initial value of 1 with probability p, or 0 with probability $1 - p$. Let

G_p be the (random) directed graph on the n nodes with an edge from u to v if and only if u sent a message to v and the message was sent before v sent any message to u as $\mathcal{A}$ executed from the starting configuration C_p. We now show that with probability at least $1 - \epsilon$, G_p is a forest comprising trees that are directed away from their respective root."

Lemma 2.8. *With probability at least $1 - \epsilon'$ for some arbitrarily small but constant $\epsilon' > 0$, the graph G_p for all $p \in [0, 1]$ is a forest in which each tree contains exactly one node (called its root) with zero in-edges and, furthermore, at least two tree edges are oriented away from the root.*

Proof. Recall that the number of messages passed by $\mathcal{A}$ is at most $o(n^{1/2}/\alpha^{3/2})$ with probability at least $1-o(1)$. Therefore, with probability at least $1-o(1)$, we can apply the condition that the number of nodes that participate in any form of communication (either sending or receiving) is at most $o(n^{1/2}/\alpha^{3/2})$. Recall also that each message is sent to a random node. From the Lemma 2.4, in a similar way, there exist at least $1/2\alpha$ initiator nodes. Therefore, the smallest tree possess at max $\frac{o(n^{1/2}/\alpha^{3/2})}{1/2\alpha}$ i.e., $o(n^{1/2}/\alpha^{1/2})$ nodes. Also, a node is non-faulty with at least α probability. Hence, there exist $o(n^{1/2}/\alpha^{1/2})\alpha$ i.e., $o(n^{1/2}\alpha^{1/2})$ good nodes in the smallest tree in expectation. By Chernoff bound (see Theorem 4.4 in [112]), we have $7 \times o(n^{1/2}\alpha^{1/2})$ i.e., $o(n^{1/2}\alpha^{1/2})$ good nodes with high probability. Recall that in the network there are $o(n^{1/2}/\alpha^{1/2})$ good nodes in expectation with high probability. Therefore, the probability that none of those $o(n^{1/2}\alpha^{1/2})$ messages is targeted towards either a good node that has sent messages or a good node that has already received some message is at least $(1 - o(n^{1/2}\alpha^{1/2})/n)^{o(n^{1/2}/\alpha^{1/2})} \geq 1 - o(1)$. Removing the conditioning, we get the probability that at least two components in G_p are rooted and oriented trees is again at least $1 - \epsilon'$ for some fixed $\epsilon' > 0$. □

Thus, for the rest of the argument, we apply the condition that for all $p \in [0, 1]$, G_p is a forest as described in Lemma 2.8. We say that a tree is a deciding tree if there is at least one deciding node within that tree. The Lemma 2.9 and Lemma 2.10 with supporting details are highly adapted from the work done in [14]. Informally, the arguments work for the non-faulty settings' lower bounds are also adaptable to the faulty settings.

Lemma 2.9. *"For every value of $p \in [0, 1]$, the probability that there are at least two deciding trees is at least a constant.*

Proof. Since $\mathcal{A}$ reaches agreement with probability at least $1-\epsilon$, the probability that no tree decides is at most ϵ.

For all values of p, the probability that there is at most one deciding tree is at most a suitable constant c bounded away from 1. Otherwise, there exists a p value for which, with probability at

least c, exactly one tree decides. Then, quite easily, the root of that deciding tree can be made a leader. This yields a leader election protocol that succeeds with probability at least c, which will contradict Theorem 2.3 when c is a sufficiently large constant strictly less than $1 - \epsilon$.

Adding up all these possibilities, we are still left with a probability of at least a constant, $q < 1 - \epsilon - c$ with which there must be two or more deciding trees. □

Now we know that there are at least two deciding trees with constant probability, we show that these trees reach opposing decisions with probability bounded from below by a constant, thereby leading to a contradiction that $\mathcal{A}$ reaches agreement with high probability.

Lemma 2.10. *There exists a value $p^* \in [0, 1]$ such that the probability that there are at least two deciding trees in G_{p^*} with opposing decisions is at least a constant.*

Proof. We can define the probabilistic valency of p with respect to $\mathcal{A}$, denoted V_p, as the probability that $\mathcal{A}$ will terminate with a decision value 1 under the initial configuration C_p. Thus, the probabilistic valencies of $p = 0$ and $p = 1$ are 0 and 1, respectively. It is easy to see that V_p is a continuous function of $p \in [0, 1]$ as infinitesimally small changes to p can only result in infinitesimally small changes to V_p. This is true because infinitesimally small changes to p will only result in infinitesimally small changes to C_p, and since the algorithm's behavior is based on the initial configuration (which does not change much), V_p will also only change infinitesimally. This continuity in V_p implies that for every $x \in [0, 1]$, there is a p such that $V_p = x$.

We prove this lemma under the condition that there are at least two deciding trees and that these two trees do not interact with each other. We have seen that enforcing this conditioning requires a probability of q bounded away from 0. An important implication here is that, despite this conditioning, the probabilities of the two decision values must be bounded from below by a constant. For the sake of contradiction, let us suppose (without loss of generality) that the algorithm decides 1 with probability at most $o(1)$ under this conditioning. Let us fix p such that $V_p = 1 - q/2$. Then, the probability that it decides 1 without the conditioning will be at most $q(o(1)) + (1 - q) < 1 - q/2$, a contradiction.

Now consider two deciding trees T_1 and T_2. For simplicity, let us assume that the decision outcome of T_1 is (say) 0; this occurs with some constant probability. We now reason that the probability that the decision outcome of T_2 is 1 will be at least a constant. This is true because the random input values for T_2 were all chosen independently of the input values for T_1. Thus, the two trees can decide contradictory values with probability at least a constant. □

Thus, we have shown that when the nodes are anonymous and the number of messages sent is at most $o(n^{1/2}/\alpha^{3/2})$, implicit agreement is not reached with probability at least a constant. To

generalize to the case where nodes do have IDs, we simply assume that the adversary provides ID's chosen uniformly at random from, say, $[1, n^4]$. Let $\mathcal{A}^*$ be an algorithm that exploits these unique IDs.

Notice, however, that the execution of $\mathcal{A}^*$ that exploits the adversarially generated random IDs is essentially akin to the execution of $\mathcal{A}^*$ in the anonymous setting in which each node first generates a random number between $[1, n^4]$ and then uses that random number as their IDs. The only difference is that multiple nodes may generate the same random number, thereby inheriting the same ID, but this is an event whose probability is at most $1/n$, so when we condition on these random numbers being distinct, the two executions are probabilistically identical, so any lower bound on the execution without adversarially generated IDs will extend to the case where nodes have unique IDs provided by the adversary." Thus,

Theorem 2.7. *Suppose $\mathcal{A}$ is an algorithm that solves implicit agreement with probability at least $1 - \epsilon$ for some sufficiently small constant $\epsilon > 0$ in a network of n nodes with unique IDs. Then, with probability at least a constant, the message complexity of $\mathcal{A}$ is at least $\Omega(n^{1/2}/\alpha^{3/2})$.*

2.6 Conclusion

In this chapter, we studied the role played by randomization in the fault-tolerant distributed network. In particular, we examined how randomization can help to solve agreement and leader election efficiently in a crash-fault setting. We showed that if a constant fraction of nodes is faulty, then the message complexity of leader election and agreement is asymptotically the same as in the fault-free network (complete network). We also showed a non-trivial lower bound of message complexity of both problems.

The work of this chapter open ups several interesting research problems. There is a gap between the upper and lower bounds of the message complexity of leader election. Is it possible to narrow down this gap? The other interesting question raised by our complete graph is to extend the study of the message complexity of the problem in diameter two graphs and general graphs. Finally, is it possible to provide more power to the adversary and have similar results, i.e., whether a sublinear message bound agreement protocol is possible in the presence of Byzantine node failure?

Chapter 3

Sublinear Message Bounds for Authenticated Byzantine Agreement

In this chapter, we examine the message complexity of authenticated Byzantine agreement (BA) in fully-connected, synchronous distributed networks with an honest majority [1]. Our focus is on the "implicit" version of the Byzantine agreement problem, where each node begins with an input value, and at the end, a non-empty subset of honest nodes must agree on a common input value while meeting the BA properties (even if some nodes remain undecided). It is worth noting that implicit BA is an extension of the classical BA problem.

We first show that a sublinear (in n, number of nodes) message complexity BA protocol under honest majority is possible in the standard PKI model when the nodes have access to an unbiased global coin and hash function. We further extend the result to Byzantine subset agreement, where a non-empty subset of nodes should agree on a common value. We analyze several relevant results which follow from the construction of the main result.

Further, we present a randomized Byzantine agreement algorithm which, with high probability achieves implicit agreement, uses $\tilde{O}(\sqrt{n})$ messages and runs in $\tilde{O}(1)$ rounds while tolerating $(1/2 - \epsilon)n$ Byzantine nodes for any fixed $\epsilon > 0$, the notation $\tilde{O}$ hides a $O(\operatorname{polylog} n)$ factor. The algorithm requires a standard cryptographic setup PKI and hash function with a static Byzantine adversary. The algorithm works in the CONGEST model and each node does not need to know the identity of its neighbors, i.e., works in the KT_0 model. The message complexity (and also the time complexity) of our algorithm is optimal up to a $\operatorname{polylog} n$ factor, as we show a $\Omega(\sqrt{n})$ lower bound on the message complexity. We also perform experimental evaluations and highlight the effectiveness and efficiency of our algorithm. The experimental results outperform the theoretical guarantees.

[1] This chapter is based on joint work with Anisur Rahaman Molla and contains the material from [93].

3.1 Introduction

Byzantine agreement is a fundamental and long-studied problem in distributed networks [100, 11, 106]. In this problem, all the nodes are initiated with an input value. The Byzantine agreement problem is required to satisfy: (i) the honest nodes must decide on the same input value; and (ii) if all the honest nodes receive the same input value, then they must decide on that value[2]. This should be done in the presence of a constant fraction of Byzantine nodes that can arbitrarily deviate from the protocol executed by the honest nodes. Byzantine agreement provides a critical building block for creating attack-resistant distributed systems. Its importance can be seen from widespread and continued application in many domains such as wireless networks [81, 103, 134, 129], sensor networks [128], grid computing [8], peer-to-peer networks [124] and cloud computing [135], cryptocurrencies [26, 44, 3, 80, 109], secure multi-party computation [24] etc. However, despite huge research, we lack efficient practical solutions to the Byzantine agreement for large networks. A main drawback for this is the *large message complexity* of currently known protocols, as mentioned by many systems papers [5, 7, 28, 108, 137]. The best-known Byzantine protocols have (at least) quadratic message complexity [61, 25, 45, 89], even in the authenticated settings [40, 113]. In a distributed network, nodes communicate with their neighbors by passing messages. Therefore, communication cost plays an important role to analyze the performance of the algorithms, as also mentioned in many papers [1, 65, 63, 113].

King and Saia [77] presented the first Byzantine agreement algorithm that *breaks* the quadratic message barrier in synchronous, complete networks. The message complexity of their algorithm is $\tilde{O}(n^{1.5})$. Later, Braud-Santoni et al. [27] improved this to $\tilde{O}(n)$ message complexity. Both works require the nodes to know the IDs of the other nodes a priori. This model is known as KT_1 model (knowledge till hop 1) [118]. Another challenging model is KT_0, where nodes do not know their neighbors a priori [118]. Note that in KT_0 model, nodes can know their neighbors easily by communicating to all the neighbors, perhaps in a single round, but that will incur $\Omega(n^2)$ messages. The KT_0 model is more appropriate to the modern distributed networks which are permissionless, i.e., nodes can enter and leave the network at will.

In this chapter, our main focus is to study the message complexity of the Byzantine agreement problem in the KT_0 model under the assumption of cryptographic setup and a global coin (as defined in [119]). In fact, we study the implicit version of the Byzantine agreement, where not all the honest nodes need to be decided; only a non-empty subset of the honest node must decide on

[2]Throughout, we interchangeably use the term 'non-Byzantine' and 'honest', and similarly, 'Byzantine' and 'faulty'.

an input value. Our main result is a randomized algorithm to solve implicit Byzantine agreement using sublinear messages (only $\tilde{O}\sqrt{n}$) while tolerating $f \leqslant (1/2 - \epsilon)n$ Byzantine nodes, where n is the number of nodes in the network, f is the number of Byzantine nodes and $\epsilon > 0$ is a fixed constant. The implicit algorithm can be easily extended to the explicit Byzantine agreement (where all the honest nodes must decide) using $O(n \log n)$ messages only. The algorithm is simple and easily implementable, which is highly desired for practical purposes. While the assumptions on the Public Key Infrastructure (PKI) set up with keyed hash function and the global coin together make the model a little weaker, they are realistic and implementable [3]. Similar assumptions were made earlier in the literature, e.g., in Algorand, Gilad et al. [58] uses "seed" and PKI setup, where "seed" is essentially the shared random bits. In their approach, they formed a set of candidate nodes and reached an agreement with the help of the sortition algorithm (see Section 5 of [58]) using proof-of-stake (PoS). They further used verifiable random functions (VRFs) [110] for the verification of the candidate nodes which return hash and proof. In our work, we do not require the assumption of VRFs. The message complexity of Algorand is $\tilde{O}(n)$, albeit for the explicit agreement.

Without PKI setup, hash function and global coin assumptions, we do not know if a sublinear (or even a linear) message complexity Byzantine agreement algorithm is possible or not. So far, the best results have quadratic message bound in the KT_0 model and sub-quadratic in the KT_1 model.

Our result introduces the first *sublinear message complexity* Byzantine agreement algorithm and at the same time tolerates optimal resilience, i.e., $f \leqslant (1/2 - \epsilon)n$. We further extend the implicit agreement to a natural generalized problem, called *subset agreement* problem. The subset agreement problem can be useful in real applications. For example, in a large scale distributed network, it may require that a non-empty subset of the nodes (unknown to each other) want to agree on a common value. Since, typically, the size of the subset is much smaller than the network size, the cost of the agreement would be less than the explicit agreement. Thus, subset agreement may work as a subroutine in many applications. We also argue a lower bound on the message complexity of the problem. The lower bound shows that the message complexity of our algorithm is optimal up to a $\mathrm{polylog}\, n$ factor. Finally, we implement our algorithm to evaluate its actual performance. Our results can be viewed as a step towards understanding the message complexity of randomized BA in distributed networks under the assumptions of PKI, hash function and global coin.

Chapter Organization: The rest of the chapter is organized as follows. In the rest of this section, we state our main result and introduce the model and definition. Section 3.2 is a related

[3]Throughout, we interchangeably use the term 'hash function' and 'keyed hash function'.

work, which introduces the seminal works done in the same direction. Section 2.5 presents the main implicit Byzantine agreement algorithm and other relevant results. Section 3.4 presents the Byzantine subset agreement. In Section 3.5, we present the lower bound on the message complexity to support the optimality of our algorithm. Section 3.6 shows the experimental evaluation of the implicit Byzantine agreement algorithm. Finally, we conclude in Section 3.7.

3.1.1 Our Main Results

We show the following main results.

Theorem 3.1 (Implicit Agreement)**.** *Consider a synchronous, fully-connected, anonymous network of n nodes and CONGEST communication model. Assuming a public-key infrastructure with the keyed hash function, there exists a randomized algorithm which, with the help of global coin, solves implicit Byzantine agreement with high probability in $O(\log^2 n)$ rounds and uses $O(n^{0.5}\log^{3.5} n)$ messages while tolerating $f \leqslant (1/2-\epsilon)n$ Byzantine nodes under non-adaptive adversary, where ϵ is any fixed positive constant.*

Theorem 3.2 (Explicit Agreement)**.** *Consider a synchronous, fully-connected network of n nodes and CONGEST communication model. Assuming a public-key infrastructure with the keyed hash function, there exists a randomized algorithm which, with the help of global coin, solves Byzantine agreement with high probability in $O(\log^2 n)$ rounds and uses $O(n \log n)$ messages while tolerating $f \leqslant (1/2-\epsilon)n$ Byzantine nodes under non-adaptive adversary, where ϵ is any fixed positive constant.*

Theorem 3.3 (Byzantine Subset Agreement)**.** *Consider a complete n-node network $G(V, E)$ and a subset $S \subseteq V$ of size k. There is a randomized algorithm which, with the help of a global coin and PKI set up with keyed hash function, solves the Byzantine subset agreement over S with high probability and finishes in $\tilde{O}(k)$ rounds and uses $\min\{\tilde{O}(k\sqrt{n}), \tilde{O}(n)\}$ messages.*

Theorem 3.4 (Lower Bound)**.** *Consider any algorithm A that has access to an unbiased global coin and sends at most $f(n)$ messages (of arbitrary size) with high probability on a complete network of n nodes. If A solves the authenticated Byzantine agreement under honest majority with constant probability, then $f(n) \in \Omega(\sqrt{n})$.*

Finally, we implement our implicit agreement algorithm and show its effectiveness and efficiency for different sizes of Byzantine nodes. When the Byzantine nodes behave randomly, the experimental results perform better than the theoretical guarantees.

3.1.2 Model and Definitions

The network is a synchronous and fully connected graph of n nodes. Initially, nodes do not know their neighbors, also known as KT_0 model [118]. Nodes have access to an unbiased global coin through which they can generate shared random bits. The network is $f \leq (1/2 - \epsilon)n$ resilient, i.e., at most $(1/2 - \epsilon)n$ nodes (among n nodes) could be faulty, for any constant $\epsilon > 0$. We consider Byzantine fault [100]. A Byzantine faulty node can behave maliciously such that it sends any arbitrary message or no message in any round to mislead the protocol, e.g., it may send different input values to different nodes, or it may not send any message to some of the nodes in a particular round. We assume that a *static* adversary controls the Byzantine nodes, which selects the faulty nodes before the execution starts. However, the adversary can adaptively choose when and how a node behaves maliciously. Further, the adversary is rushing and has full information – the adversary knows the states of all the nodes and can view all the messages in a round before sending out its own messages for that round. We assume that each node possesses multi-valued input of size $O(\log n)$ provided by the adversary.

We assume the existence of digital signatures, Public Key Infrastructure (PKI) and hash function. Each node possesses a public-secret key pair (p_k, s_k). Secret keys are generated randomly, and correspondingly public keys are generated with the help of a prime order group's generator. Therefore, the public keys are not skewed but random. Trusted authority also provides other cryptographic primitives for each node, and certifies each node's public keys. Nodes use digital signatures for the authentication of any information. We abstract away the details of the cryptographic setup; assuming it is a standard framework. Public key (p_k) of all the nodes, hash function, shared random bits (generated through global coin) and n are the common knowledge for all the nodes.

We consider the *CONGEST* communication model [118], where a node is allowed to send a message of size (typically) $O(\log n)$ or $O(\text{polylog}(n))$ bits through an edge per round. The message complexity of an algorithm is the total number of messages sent by all the non-faulty nodes throughout the execution of the algorithm.

Definition 3.1 (Implicit Byzantine Agreement)**.** *Suppose initially all the nodes have an input value (say, provided by an adversary). An implicit Byzantine agreement holds when the following properties hold: (i) the final state of all the non-Byzantine nodes is either "decided" or "undecided"; (ii) all the "decided" non-Byzantine nodes must agree on the same value (consistency property); (iii) if all the non-Byzantine nodes have the same input value then they must decide on that value (validity property); (iv) all the non-Byzantine nodes eventually reach to the final state, where at*

least one non-Byzantine node must be in the "decided" state (termination).

Definition 3.2 (Byzantine Subset Agreement)**.** *Suppose initially all the nodes have an input value (say, provided by an adversary). The Byzantine subset agreement is an agreement by a (specified) non-empty subset $S \subseteq V$ of nodes such that the fraction of Byzantine nodes in S is preserved, i.e., $f_S \leqslant (1/2 - \epsilon)|S|$, where f_S is the number of Byzantine nodes in S. We assume that each node knows whether it belongs to S or not, but does not know the identities of the other nodes in the subset. The agreement on S holds when the final state of all the non-faulty nodes in S is "decided" and the deciding value of all of them follows the properties: (i) all the non-faulty nodes must decide on the same value (consistency property); (ii) if all the non-faulty nodes of network have the same initial value then they (the non-faulty nodes in S) must decide on that value (validity property); (iii) all the non-faulty nodes (of S) eventually decide on a value (termination property).*

Definition 3.3 (Keyed Hash Function)**.** *Let $\mathcal{K}_h, X$ be two non-empty finite sets and H be an n-bit function such that $H : \mathcal{K}_h \times X \rightarrow \{0,1\}^n$, where H follows three properties: (i) Preimage Resistant – Given a hash value h, it should be difficult to find any message m such that $h = H(k, m)$. (ii) Second Preimage Resistant – Given an input m_1, it should be difficult to find a different input m_2 such that $H(k, m_1) = H(k, m_2)$. (iii) Collision Resistant – It should be difficult to find two different messages m_1 and m_2 such that $H(k, m_1) = H(k, m_2)$.*

3.1.3 Byzantine Agreement vs. Byzantine Broadcast

Byzantine Agreement is typically studied in two forms while they are equivalent, i.e., one can be reduced to the other [100]. In the agreement version, also known as *Byzantine Consensus*, prior to the protocol starts, all the nodes receive an input value. Byzantine agreement is achieved if the following three properties hold.
Consistency: all of non-faulty nodes must decide on the same value.
Validity: if all the non-faulty nodes have the same initial value, then they must decide on that value.
Termination: all the non-faulty nodes eventually decide on a value.

In the *Byzantine Broadcast*, there is a designated sender (could be Byzantine or honest) who broadcasts the input values to the nodes. Termination and consistency are the same as in the case of Byzantine agreement. In the case of validity, all the non-faulty nodes output the same value if the sender is honest. Also, that value should be the input value of the sender.

3.2 Related Work

Byzantine agreement and Byzantine broadcast have been studied extensively in various models and settings in the last four decades, starting from its classic introduction by Lamport, Shostak and Pease [100, 116]. They presented protocols and fault tolerance bounds for two settings (both synchronous). Without cryptographic assumptions (the unauthenticated setting), Byzantine broadcast and agreement can be solved if $f < n/3$. Assuming digital signatures (the authenticated setting), Byzantine broadcast can be solved if $f < n$ and Byzantine agreement can be solved if $f < n/2$. The initial protocols had exponential message complexities [100, 116]. Fully polynomial protocols were later shown for both the authenticated ($f < n/2$) [40] and the unauthenticated ($f < n/3$) [55] settings. Both protocols require $f + 1$ rounds of communication, which matches the lower bound on round complexity for deterministic protocols [48].

Our Byzantine agreement algorithm makes use of a few past results. First, we make use of the concept of a set of candidate nodes, which is a subset of nodes. The notion of a committee (set of candidate nodes) is used in, e.g., [14, 59, 92, 97]. Finally, we adapt the Byzantine Agreement algorithm designed by Dolev et al. [40].

The previous results in the same direction are summarized in Table 3.1. Santoni et al. [27] achieved the $\tilde{O}(n)$ communication complexity against a non-adaptive adversary while resilience is $f < n/(3+\epsilon)$ and rushing adversary with KT_0 model. The work of King-Saia [77] and Abraham et al. [2] used the shared random value in one or another way. King-Saia reached almost everywhere to everywhere agreement in communication complexity (communication complexity of a protocol is the maximum number of bits sent by all the non-Byzantine nodes combined across all executions) $\tilde{O}(n^{1.5})$ bits with adaptive security while resilience is $f < (1/3 - \epsilon)n$ and in the KT_1 model. Abraham et al. showed the quadratic communication complexity in expectation with an adaptive adversary while using the cryptographic assumptions and tolerated $f < n/2$ Byzantine nodes. Dolev-Strong [40] achieved cubic communication complexity with an adaptive adversary and cryptographic assumptions by tolerating $f < n/2$ Byzantine nodes. Later, Momose-Ren [113] improved the communication complexity to quadratic communication. In comparison with our work, we are using shared random value and non-adaptive adversary with rushing adversary. Shared random value help us to break the linear communication (message) complexity. In implicit agreement, we have sublinear communication complexity while in explicit agreement it is linear with cryptographic assumptions and tolerates $f \leq (1/2 - \epsilon)n$ Byzantine nodes.

Comparison of the Results					
Protocol	Agreement Type	Communication (in bits)	Adversary	Cryptographic Assumptions	Resilience
Santoni et al. [27]	Implicit	$\tilde{O}(n)$	Non-adaptive	No	$f < n/(3+\epsilon)$
King-Saia [77]	Explicit	$\tilde{O}(n^{1.5})$	Adaptive	No	$f \leq (1/3-\epsilon)n$
Dolev-Strong [40]	Explicit	$\tilde{O}(n^3)$	Adaptive	Yes	$f < n/2$
Momose-Ren [113]	Explicit	$\tilde{O}(n^2)$	Adaptive	Yes	$f < n/2$
Abraham et al. [2]	Explicit	$O(n^2)^*$	Adaptive	Yes	$f < n/2$
This chapter	Implicit	$\tilde{O}(n^{0.5})$	Non-adaptive	Yes	$f \leq (1/2-\epsilon)n$
This chapter	Explicit	$\tilde{O}(n)$	Non-adaptive	Yes	$f \leq (1/2-\epsilon)n$

Table 3.1: Comparison of various models with our result. ϵ is any positive constant. Our results assume a global coin and hash function while others are not. * indicates the bound holds in expectation.

3.3 Authenticated Implicit Byzantine Agreement

In this section, we present a randomized Byzantine agreement algorithm in a complete n-node network that tolerates $f \leq (1/2-\epsilon)n$ Byzantine nodes under a public-key infrastructure, keyed hash function and access to a global coin. The algorithm incurs $\tilde{O}(\sqrt{n})$ messages, has latency $\tilde{O}(1)$, and has high success probability.

In the algorithm, we run a subroutine BA protocol, which can tolerate $f \leq (1/2-\epsilon)n$ Byzantine nodes and may have a polynomial message (and time) complexity. In fact, we adapt the classical

algorithm presented by Dolev-Strong [40].[4] Dolev-Strong designed an algorithm for the Byzantine broadcast (BB) problem, which can be converted into a Byzantine agreement algorithm with an initial round to broadcast the input Byzantine nodes under a public-key infrastructure and access to an unbiased global coin. The communication/bit complexity is $O(\kappa n^3)$ [49]. However, using multi-signature it can be improved to $O(\kappa n^2 + n^3)$, where κ is a security parameter which is essentially the maximum size of the messages [113]. While the original Dolev-Strong BB protocol tolerates $f \leqslant n-1$ faults, the converted BA protocol works for the honest majority nodes, i.e., tolerates $f < n/2$ Byzantine faults which is optimal for an authenticated BA [2, 46, 73, 100, 111, 113].

Dolev-Strong algorithm is deterministic and has a latency of $f+1$ rounds. The general idea of the BB protocol is to form a signature chain consisting of signatures from distinct nodes. A signature chain of $f+1$ signatures must contain a non-faulty signature from a non-faulty node which can send the value to all the other nodes. The protocol is designed in such a way that in $f+1$ rounds it forms a signature chain of size $f+1$. We adapted the Dolev-Strong BB protocol for the Byzantine agreement problem and used it in our implicit BA algorithm.

Let us now describe the implicit BA algorithm. The public key (p_k) of the nodes is known to all the nodes (as distributed by the trusted third party), but a node does not know which port or edge is connecting to which node (having a particular public key). To minimize the message complexity, a generic idea is to select a set of small-size candidate nodes, which will be responsible for solving the (implicit) agreement among themselves. Thus, it is important to have the honest majority in the set of candidate nodes. For this, a random set of nodes, called *candidate nodes* or *committee nodes*, of size $O(\log n)$ is selected. Let us denote the candidate nodes set by $\mathcal{C}$. A Byzantine node may try to claim that it is in $\mathcal{C}$, which needs to be stopped to guarantee the honest majority in $\mathcal{C}$. To overcome this problem, we take the help of a global coin and a keyed hash function, which together determine the candidate nodes.

A common random number, say r, is generated with the help of the global coin. The random number should be large enough to use as the key to the hash function. Note that the random number is generated after the selection of the Byzantine nodes by the adversary. Every node uses its respective public key (p_k) as the message and r as the key of the hash function, say, H. That is they compute $H_r(p_k)$. For each p_{k_i}, we represent its hash value $H_r(p_{k_i})$ as H_i. Since the hash values are random (with high probability), the smallest $c \log n$ values among the n hash values are chosen to be the candidate nodes, where c is a suitable constant (to be fixed later). More precisely, a node i with the hash value H_i is in $\mathcal{C}$ if it is in the smallest $c \log n$ values of the set $\{H_i : i = 1, 2, \ldots, n\}$. Therefore, a node can easily figure out the candidate nodes since it knows

[4] One can use other suitable BA protocols, e.g., the protocol in [113].

the random number r, the hash function, and the public keys of all the nodes. However, the node does not know the ports connecting to them (as KT_0 model).

Although a node knows all the candidate nodes (in fact, their public keys), it does not know the edges connecting to the candidate nodes, since the network is anonymous, i.e., KT_0 model. It can be known by the candidate nodes by sending a message to all the nodes, but that will cost $n \log n$ messages. Since knowing each other is message expensive in this model, the candidate nodes communicate among themselves via some other nodes. For this, each candidate node randomly samples $\Theta(\sqrt{n \log n})$ nodes among all the n nodes; call them as *referee nodes*. The reason behind sampling so many referee nodes is to make sure at least one common "non-faulty" referee node between any pair of candidate nodes. The candidate nodes communicate with each other via the referee nodes. Notice that a node may be sampled as a referee node by multiple candidate nodes. It might happen that a Byzantine referee node may change the value of an honest candidate node before forwarding it to the candidate nodes. To avoid this, we take advantage of digital signatures. Each referee node signs the input value before transmitting it to its referee nodes. As a digital signature helps to detect forging messages, a candidate node considers only the genuine messages received from the referee nodes.

Thus, we have a small committee of nodes (i.e., $\mathcal{C}$) with the honest majority and the committee nodes can communicate via the referee nodes. Then we apply the Dolev-Strong BA protocol [40] in the committee to achieve agreement. Let us now present the adapted Dolev-Strong algorithm to work for the Byzantine agreement.

Step 0: Each candidate node u does the following in parallel. u signs and sends its input value to its corresponding referee nodes, say, $\mathcal{R}_u$. Each referee node w sends all the received values to its respective candidate nodes, say, $\mathcal{C}_w$. Therefore, the candidate nodes have the input values of all the candidate nodes. Now each candidate node proposes a value for the agreement based on the priority, along with all the signatures received corresponding to that input value. If there is a value that is sent by the majority of the nodes (i.e., more than $(c \log n)/2$ nodes), then that value gets the *highest priority*. In case of more than one majority, the value proposed by the maximum number of nodes gets the highest priority. There might be the case, two values are proposed by the same number of nodes; in that case, the larger input value gets the highest priority. If a candidate node has the highest priority input value, then it sends the value (after signing) to all the candidate nodes along with the received signatures for the highest priority value. Otherwise, the candidate node does not send anything. The reason of getting more than one majority value is that a Byzantine node may propose different values to different nodes. If no such majority value is received, then the candidate nodes decide on a default value, say, the minimum in the input value set. By sending

the input value, we mean sending the input value along with all the signatures corresponding to that input value.

Then, the following two steps are performed iteratively[5].

Step 1: If a candidate node u receives a set of at least i legitimate signatures in i^{th} iteration with the highest priority value than its earlier sent value, then u proposes this new highest priority value along with all the signatures to its referees nodes $\mathcal{R}_u$.

Step 2: If a referee node w receives a set of at least i legitimate signatures in i^{th} iteration (with the highest priority value) then w forwards this highest priority value to its corresponding candidate nodes $\mathcal{C}_w$.

The above two steps (i.e., Step 1 and 2) are performed for $O(c \log n)$ iterations and then the algorithm terminates. In the end, all the (non-faulty) candidate nodes have the same value, either the default value or the value possessed by the majority of nodes (the highest priority value). Intuitively, there are $O(c \log n)$ candidate nodes and a single node may propose the highest priority value in each iteration (say, the single node with the highest priority value is faulty and send to only faulty nodes) and thus the highest priority value may propagate slowly. But eventually, the highest priority value is received by all the candidate nodes as the set of candidate nodes contains the majority of the non-faulty nodes (see Lemma 3.1) and there is a common non-faulty referee node between any pair of candidate nodes (see Lemma 3.2). On the other hand, if there is no highest priority value, then they decide on the default value. Thus, the (honest) candidate nodes agree on a unique value. A pseudocode is given in Algorithm 3.

Let us now show the above claims formally. We first show that the majority of the nodes in the candidate set are honest.

Lemma 3.1. *The number of Byzantine nodes in the candidate set $\mathcal{C}$ is strictly less than $\frac{1}{2}|\mathcal{C}|$.*

Proof. Let α fraction of the nodes in the network are Byzantine where $\alpha = 1/2 - \epsilon$ is a fixed constant. So $\alpha + \epsilon = 1/2$. The nodes in $\mathcal{C}$ are chosen uniformly at random (i.e., with probability $1/n$) and the number of Byzantine nodes is at most αn, the probability that a particular node in $\mathcal{C}$ is Byzantine is at most α . Let $\mathcal{C}$ contain k nodes, $\{u_i \mid i = 1, 2, \ldots, k\}$. Let us define random variables X_is such that $X_i = 1$ if u_i is Byzantine, and 0 otherwise. Further, $X = \sum_{i=1}^{k} X_i$ is the total number of Byzantine nodes in $\mathcal{C}$. Then, by linearity of expectation, $E[X] \leqslant \alpha k$. Then, by Chernoff bounds [112],

[5] An iteration is the number of rounds required to send messages from one candidate node to all the candidate nodes via the referee nodes. Since we consider the CONGEST model, an iteration may take up to $O(\log n)$ rounds in our algorithm.

Algorithm 3 AUTHENTICATED-IMPLICIT-BA

Require: A complete n node anonymous network with $f \leqslant (1/2 - \epsilon)n$ Byzantine nodes. Each node receives an input value provided by an (static) adversary, a pair of public-private keys (p_k, s_k), keyed hash function and a global coin. $\epsilon > 0$ is a fixed constant.

Ensure: Implicit Agreement.

1: Select $c \log n$ nodes as the candidate nodes set (say, $\mathcal{C}$), which have the smallest $c \log n$ hash values generated with the help of public key and random number. The value of the constant c is $3\alpha/\epsilon^2$, follows from Lemma 3.1.
2: Each candidate node u randomly samples $2\sqrt{n \log n}$ nodes as referee nodes (say, $\mathcal{R}_u$).
3: Each candidate node u signs and sends its input value (with signature) to $\mathcal{R}_u$.
4: Each referee node w sends all the received values to their respective candidate nodes $\mathcal{C}_w$ along with the legitimate signatures one by one. It takes $O(\log n)$ rounds.
5: Each candidate node u sends the input value along with all the received signatures to $\mathcal{R}_u$ based on the (highest) priority. ▷ Highest priority is defined in the description, Step 0.
6: **for** the next $(c \log n)$ iterations, the candidate and referee nodes in parallel **do**
7: Each referee node w checks:
8: **if** w receives a set of at least i legitimate signatures in the i^{th} iteration **then**
9: w sends highest priority value to $\mathcal{C}_w$.
10: **else**
11: w does not send any messages.
12: **end if**
13: Each candidate node u checks:
14: **if** u receives a set of at least i legitimate signatures in the i^{th} iteration with a highest priority value than it sent earlier **then**
15: u sends highest priority value to $\mathcal{R}_u$.
16: **else**
17: u does not send any messages.
18: **end if**
19: **end for**
20: All the (non-faulty) candidate nodes have the same highest priority value on which they agree. Otherwise, if they do not receive any highest priority value, they agree on a default value, say, the minimum in the input value set.

$$\begin{aligned}\Pr(X \geqslant (\alpha+\epsilon)k) = \Pr(X \geqslant (1+\epsilon/\alpha)E[X]) &\leqslant \exp\left(-E[X](\epsilon/\alpha)^2/3\right) \\ &\leqslant \exp\left(-(k\epsilon^2)/(3\alpha)\right) \text{ for } k = |\mathcal{C}| = (3\alpha/\epsilon^2)\log n\end{aligned}$$

Thus, $\Pr(X < (\alpha+\epsilon)|\mathcal{C}|) = \Pr(X < \frac{1}{2}|\mathcal{C}|) > 1 - 1/n$. In other words, $\mathcal{C}$ contains at most

$(1/2 - \delta)|\mathcal{C}|$ Byzantine nodes with high probability, for any fixed $\delta > 0$. □

The candidate nodes communicate with each other via the referee nodes, which are sampled randomly by the candidate nodes. The number of referee nodes is sampled in such a way that there must be a common referee node between every pair of candidate nodes (so that the candidate nodes can communicate) and at the same time keep the message complexity lower. In fact, we need to guarantee a stronger result. Namely, there must be a *non-faulty* common referee node for reliable communication.

Lemma 3.2. *Any pair of candidate nodes have at least one common non-faulty referee node with high probability.*

Proof. Let us consider two candidate nodes v and w, and let x_i be the i^{th} node selected by v. Let the random variable X_i be 1 if x_i is also chosen by w and 0 otherwise. Since the x_is are chosen independently at random, the X_is are independent. So, $Pr[X_i = 1] = \frac{2\sqrt{n \log n}}{n}$. Hence, the expected number of (choice of) referee nodes that v and w haven in common are:

$$\begin{aligned} E[X] &= E[X_1] + E[X_2] + \cdots + E[X_{(2\sqrt{n \log n})}] (\textit{by linearity}) \\ &= 2\sqrt{n \log n} \cdot \frac{2\sqrt{n \log n}}{n} = 4 \log n \end{aligned} \tag{3.1}$$

Thus, by using the Chernoff bound [112], $\Pr[X < (1 - \delta)E[X]] < e^{-\delta^2 E[x]/2}$ for $\delta = 0.5$, we get

$$\Pr[X < (1 - 0.5)4 \log n] < e^{-(0.5)^2 (4 \log n)/2} < \frac{1}{\sqrt{n}} \tag{3.2}$$

For that reason at least $2 \log n$ choice of nodes by v are jointly chosen by v and w. As a consequence, the probability that none of these choices is non-faulty:

$$\left(\frac{1}{2} - \epsilon\right)^{2 \log n} < \frac{1}{n} \tag{3.3}$$

Therefore, the probability of selecting at least one non-faulty referee node from the sampled nodes of u is at least $1 - 1/n$.

□

Lemma 3.1 ensures that a committee (i.e., the candidate set) of size $O(\log n)$ with honest majority can be selected. Lemma 3.2 ensures that the committee nodes can communicate with each other reliably via the referee nodes. Thus, the BA problem on n nodes reduces to a $O(\log n)$-size

committee nodes. Then the Dolev-Strong BA protocol ensures that the committee nodes achieve Byzantine agreement among themselves deterministically. Therefore, the algorithm (Algorithm 3) correctly solves the implicit Byzantine agreement with high probability (i.e., among the committee nodes only). The non-faulty nodes which are not selected in the committee may set their state as "undecided" immediately after the selection of the candidate nodes.

Below, we analyze the message and time complexity of the algorithm.

Lemma 3.3. *The message complexity of the authenticated implicit BA algorithm is* $O(n^{0.5}\log^{3.5} n)$.

Proof. In Step 3 and 4 of the algorithm, $O(\log n)$ candidate nodes send their input value with signature to the other candidate nodes via the $2\sqrt{n\log n}$ referee nodes. Here the size of each message is $O(\kappa+\log n)$ bits, where κ is the security parameter, the size of the signature. So these two steps uses $O(\sqrt{n\log n})\cdot O(\log n)\cdot O(\kappa+\log n) = O((\kappa+\log n)\sqrt{n\log^3 n})$ bits.

Inside the $O(\log n)$ iteration (Step 6): $O(\log n)$ candidate nodes may have at most $O(\log n)$ messages to be sent to the referee nodes for $O(\log n)$ rounds. The size of each message is $O(\kappa+\log n)$ bits. So it uses $O((\kappa+\log n)\sqrt{n\log^7 n})$ bits. Further, the same number of message bits are used when the referee nodes forward the messages to the candidate nodes. Thus, a total $O((\kappa+\log n)\sqrt{n\log^7 n})$ bits are used inside the iteration.

Hence, the total communication complexity of the algorithm is $O((\kappa+\log n)\sqrt{n\log^7 n})$ bits. Since the length of κ is typically to be the maximum size of the messages [113], it is safe to assume κ is of order $O(\log n)$ for large n. Therefore, the total number of bits used is: $O(\sqrt{n\log^9 n})$. Thus, the message complexity of the algorithm is $O(\sqrt{n\log^7 n})$, since the size of each message is $O(\log n)$[6]. □

Lemma 3.4. *The time complexity of the algorithm is* $O(\log^2 n)$ *rounds.*

Proof. There are $O(\log n)$ candidate nodes, and each may have $O(\log n)$ messages to be sent to its referee nodes in parallel. Thus, it takes $O(\log^2 n)$ rounds, since it takes one round to send a constant number of messages of size $O(\log n)$ bits. The same time bound holds when the referee nodes forward the messages to the candidate nodes. Therefore, the overall round complexity is $O(\log^2 n)$. □

Thus, we get the following result of the implicit Byzantine agreement with authentication.

Theorem 3.5 (Implicit Agreement)**.** *Consider a synchronous, fully-connected network of* n *nodes and CONGEST communication model. Assuming a public-key infrastructure with the keyed hash*

[6] Alternatively, the communication complexity is $O\left(\sqrt{n\log^9 n}\right)$ bits.

function, there exists a randomized algorithm which, with the help of a global coin, solves implicit Byzantine agreement with high probability in $O(\log^2 n)$ rounds and uses $O(n^{0.5}\log^{3.5} n)$ messages while tolerating $f \leqslant (1/2-\epsilon)n$ Byzantine nodes under non-adaptive adversary, where ϵ is any fixed positive constant.

Our implicit agreement algorithm can be easily extended to solve explicit agreement (where all the honest nodes must decide on the same value satisfying the validity condition) in one more round. After the implicit agreement, the committee nodes send the agreed value (along with the signature) to all the nodes in the network in the next round. This incurs $O(n \log n)$ messages. Then all the nodes decide on the majority value since the majority of the nodes in the committee are honest. Thus, the following result of explicit agreement follows immediately.

Theorem 3.6 (Explicit Agreement)**.** *Consider a synchronous, fully-connected network of n nodes and CONGEST communication model. Assuming a public-key infrastructure with the keyed hash function, there exists a randomized algorithm which, with the help of global coin, solves Byzantine agreement with high probability in $O(\log^2 n)$ rounds and uses $O(n \log n)$ messages while tolerating $f \leqslant (1/2-\epsilon)n$ Byzantine nodes under non-adaptive adversary, where ϵ is any fixed positive constant.*

Let us now discuss some relevant follow-up results.

3.3.1 Byzantine Leader Election

A leader can be elected without any communication in this model. Since the public keys are known to all the nodes, the node nearest to the random number generated through the global coin will be the leader. In case of a tie, the node with the largest public key will be the leader. The elected leader is non-faulty with probability at most $(1 - f/n)$, where f is the number of faulty nodes. The reason is that the Byzantine nodes can act as an honest node and the probability of electing a Byzantine node as leader is the same as for an honest node. Since there are at most $f \leqslant (1/2-\epsilon)n$ Byzantine nodes, the elected leader is non-faulty with at most constant probability.

We remark that a leader solves the implicit agreement, as the leader can agree on its own input value. It also solves explicit simply by sending its value to all the nodes. However, the success probability of the Byzantine agreement is at most constant. A Byzantine agreement protocol with a high success probability is important for practical solutions (which is presented in the above section).

3.3.2 In the KT_1 Model

In the KT_1 (**K**nown **T**ill **1**) communication model, an implicit Byzantine agreement can be solved in $O(\log^2 n)$ rounds using $O(\log^3 n)$ messages in the same settings. First, select the candidate nodes of size $\Theta(\log n)$ as we selected in the KT_0 model (see Section 2.5). In the KT_1 model, each node is aware about the IDs' of its neighbor. Therefore, the candidate nodes know each other and the port connecting to them. In fact, they form a complete graph of size $\Theta(\log n)$ where one node knows the other. Now, we can run the Algorithm 3 on this complete graph of candidate nodes, which essentially solves implicit Byzantine agreement in $O(\log^2 n)$ rounds and uses $O(\log^3 n)$ messages.

The leader election can be solved in the same way as in Section 3.3.1; there is no need of any communication.

3.3.3 Removing the Global Coin and Hash Function Assumption

The assumption on accessing a global coin and hash function can be replaced by some other assumption (which might be stronger in some context). Previously, with the help of global coin, a random set of candidate nodes is selected. Recall that the access of the global coin is given to the nodes after the adversary selects the Byzantine nodes (and it is a static adversary). Otherwise, the adversary can know which nodes would be in the candidate set and based on that selects the Byzantine nodes to be in the candidate set. Suppose there is no access of global coin (or shared random bits) and hash function. Then we need to make the adversary first selects the Byzantine nodes, and then the PKI setup is imposed in the network. Since the trusted third party generates a random pair of public-secret keys (p_k, s_k), the $O(\log n)$ nodes with the smallest public keys can be taken as the candidate nodes. Thus, the randomness in the candidate nodes set is preserved as required.

3.4 Byzantine Subset Agreement

Let us suppose we have a subset S of k nodes (among the n nodes in a complete network) that want to agree on a common value. Assume the majority of the nodes in S are honest. The nodes in S do not know each other. Then one can solve the Byzantine subset agreement (see Definition 3.2) easily using the implicit Byzantine agreement algorithm in Section 2.5. The algorithm is simple. All the k nodes in S act as candidate nodes and run the rest of the implicit agreement algorithm. Since the

message complexity of the implicit agreement algorithm is dominated by $O(|\mathcal{C}||\mathbb{R}|)$, where $|\mathcal{C}|$ is the number of candidate nodes and $|\mathbb{R}|$ is the number of referee nodes sampled by each candidate node. Thus, the message complexity of the Byzantine subset agreement algorithm is $\tilde{O}(k\sqrt{n})$ and it finishes in $O(k \log n)$ rounds.

However, notice that, when $k > \Omega(\sqrt{n})$, the above message bounds are more than $\Omega(n \log n)$ which is worse than a simple $O(n \log n)$ message complexity algorithm that solves explicit agreement over all the n-nodes (c.f. Theorem 3.6). This is better if $k > \sqrt{n \log n}$. Thus, if the size of S is known, and it is less than $O(\sqrt{n})$, then run the above Byzantine subset agreement algorithm. Otherwise, if the size is larger than $O(\sqrt{n})$ then run the explicit algorithm presented in Section 2.5, Theorem 3.6.

Thus, we get the following theorem when the size of subset S is known.

Theorem 3.7 (Byzantine Subset Agreement)**.** *Consider a complete n-node network $G(V, E)$ and a subset $S \subseteq V$ of size k. There is a randomized algorithm which, with the help of a global coin, hash function and PKI set up, solves the Byzantine subset agreement over S with high probability and finishes in $\tilde{O}(k)$ rounds and uses $\min\{\tilde{O}(k\sqrt{n}), \tilde{O}(n)\}$ messages.*

3.5 Lower Bound on Message Complexity

We argue a lower bound of $\Omega(\sqrt{n})$ on the number of messages required by any algorithm that solves the authenticated Byzantine agreement under honest majority with high probability. Recall that all the nodes have access to an unbiased global coin. The nodes know the IDs of the other nodes, but are unaware of the port connecting to the IDs. Also, the communication is authenticated. Our AUTHENTICATED-IMPLICIT-BA algorithm solves multi-valued agreement in polylogarithmic rounds and uses $\tilde{O}(\sqrt{n})$ messages (see, Theorem 3.5). In a non-Byzantine setting, the multi-valued agreement can be used to elect a leader by using the IDs of the nodes as the input values. Thus, any lower bound on the message complexity (and also on the time complexity) of the leader election problem also applies to the multi-valued agreement. Therefore, the $\Omega(\sqrt{n})$ lower bound shown by [14] for the leader election problem using global coin (in the non-Byzantine setting) also applies to our multi-valued Byzantine agreement with global coin. Note that the lower bound holds because of the high success probability requirement of the agreement; otherwise, the leader election algorithm discussed in Section 3.3.1 solves the agreement with zero message cost, but with only constant success probability. However, the above argument does not hold for the binary agreement, where the input values are either 0 or 1. Below, we argue for the binary case.

Let A be an algorithm that solves the authenticated Byzantine (binary) agreement with constant probability (say, more than $1/2$) under an honest majority with the access of an unbiased global coin and uses only $o(\sqrt{n})$ messages. We show a contradiction. Recall that it is a KT_0 model; so nodes do not know which edge connects to which node-ID. To achieve agreement with constant probability, nodes must communicate with the other nodes; otherwise, if the nodes try to agree locally without any communication, it is likely that there exist two nodes that agree on two different values (this can be easily shown probabilistically). On the other hand, if all the nodes try to communicate, then the message complexity of A would be $\Omega(n)$. So, only a few nodes need to initiate the process. Thus, A must pick these few initiator nodes randomly; otherwise, if picked deterministically, the Byzantine nodes can take over the initiator nodes. The initiator nodes must communicate with the nodes in the network to achieve agreement.

The initiator nodes do not know each other. In the KT_0 setting, for any two nodes to find each other with more than $1/2$ probability requires $\Omega(\sqrt{n})$ messages – follows from the following lemma.

Lemma 3.5. *In the KT_0 model, it takes $\Omega(\sqrt{n})$ messages for any two nodes to find each other with more than constant probability.*

Proof. Suppose x and y be the two nodes. Both x and y samples $f(n)$ nodes uniformly at random from all the nodes to find out each other. Let E is the event of having collision in the random samples. Then, for $f(n) < \sqrt{n}$,

$$\Pr[E] = 1 - \left(1 - \frac{f(n)}{n}\right)^{f(n)} = 1 - e^{\frac{-(f(n))^2}{n}} < 1 - 1/e = 0.63 \tag{3.4}$$

Therefore, x and y cannot find out each other with more than constant probability with $o(\sqrt{n})$ messages. □

Each initiator node samples some nodes. The initiator and the sampled nodes may exchange messages with the other nodes in the network throughout the execution of A – all these nodes form a connected sub-graph. Let us call this sub-graph as *communication graph* of the initiator node. Thus, for every initiator node, there is a communication graph. Some of them may merge and form a single communication graph. However, it is shown in [14] that if the algorithm A sends only $o(\sqrt{n})$ messages, then there exist two disjoint communication graphs w.h.p. (see, Section 2 of [14]). Since the nodes do not know each other (in KT_0), the communication graphs have similar information. The global coin also gives the same information to all the nodes. Since the communication graphs are disjoint, no information is exchanged between them. A formal proof of

this argument can be found in [97, 14].

Let H_u and H_v be the two disjoint communication graphs corresponding to the two initiators u and v respectively. We show that H_u and H_v agree with opposite decisions, i.e., if H_u decides on 0 then H_v decides on 1 and vice versa.

Lemma 3.6. *The nodes of H_u and H_v agree with opposing decisions.*

Proof. The nodes in H_u and H_v are random since the network is anonymous and no information is known before contacting a node. Thus, the same $(1/2 - \epsilon)$ fraction of Byzantine nodes (as in the original graph G) are present in both H_u and H_v in expectation. Now consider an input distribution $\mathcal{I}$, in which each node in the graph G is given an input value 0 and 1 with probability $1/2$. Then in expectation, half of the honest nodes in H_u (and also in H_v) gets 0 and the other half gets 1. We argue that the two disjoint sets of nodes in H_u and H_v decide on two different input values under the input distribution $\mathcal{I}$. Recall that the nodes in H_u have the same information as the nodes in H_v. Further, they run the same algorithm A. The Byzantine adversary (which controls the Byzantine nodes) knows the algorithm and also the input values of the honest nodes in H_u and H_v. Since the number of Byzantine nodes in each of the H_u and H_v is almost half of their size, and $0 - 1$ distribution among the honest nodes is almost 50-50, the Byzantine nodes can control the output of the agreement in the two groups — H_u and H_v. Suppose the Byzantine nodes in H_u exchange some input bits with honest nodes to agree on 0 in H_u, then the Byzantine nodes in H_v must exchange opposite bits to agree on 1 in H_v. Thus, H_u and H_v agree with opposing decisions. □

The above lemma contradicts the assumption that A solves the Byzantine (binary) agreement with only $o(\sqrt{n})$ messages. The global coin does not help, as it gives the same information to all the nodes. Thus, we get the following result on the lower bound of the message complexity.

Theorem 3.8. *Consider any algorithm A that has access to an unbiased global coin and sends at most $f(n)$ messages (of arbitrary size) with high probability on a complete network of n nodes. If A solves the authenticated Byzantine agreement under honest majority with more than $1/2$ probability, then $f(n) \in \Omega(\sqrt{n})$.*

Note that the lower bound holds for the algorithms in the *LOCAL* model, where there is no restriction on the size of a message that can be sent through edges per round [118].

3.6 Experimental Evaluation

We implement our Authenticated Implicit BA algorithm (Algorithm 3) to evaluate the performance with respect to different sizes of the Byzantine nodes. We consider the following simulation framework. We consider n nodes and $c \log n$ committee nodes, where the constant c is $3\alpha/\epsilon^2$ such that $\alpha = 1/2 - \epsilon$ measures the fraction of the Byzantine nodes, see Lemma 3.1. The binary input value is assigned randomly from the set $\{0, 1\}$ to each node. The Byzantine nodes are chosen randomly. Since it is difficult to adapt the Byzantine nodes' arbitrary behavior in the experiment, we simply consider the following random strategy of the Byzantine nodes. The Byzantine nodes randomly (with probability $1/2$) decide whether they would send a value or not in a round. If a Byzantine node decides to send, it decides randomly (with probability $1/2$) whether it will send its input value or the opposite value.

For each simulation, we consider 2^{60} to 2^{70} nodes. Each simulation is run 20 times and takes the average. The plotting of the graph is shown for the four different sets of Byzantine nodes, namely, $\sqrt{n}, n/10, n/4$ and $3n/10$. We plot the graph for a number of nodes at x-axis (in Figure 3-1 and Figure 3-2) while rounds and messages at y-axis in Figure 3-1 and Figure 3-2, respectively, by varying the parameter w.r.t number of Byzantine nodes and a number of nodes.

Summary of the round complexity: Figure 3-1 shows that as the number of Byzantine nodes increases, the number of rounds increases. This happens due to the increment of the constant value c. For example, as Byzantine nodes vary from $n/4$ to $3n/10$ then the constant c varies from 12 to 22.5. Since the number of Byzantine nodes is known, therefore, committee nodes wait for the delay imposed by the Byzantine nodes. After the delay imposed by the Byzantine nodes, honest nodes sends all the received input value from Byzantine nodes (if any). Consequently, the number of rounds is lower than the theoretical guarantees. More specifically, the experimental round complexity is $(1/2 - \epsilon)$ times, i.e., the fraction of Byzantine nodes times lower than the theoretical complexity. For example, if we consider number of nodes are 2^{60} and Byzantine nodes are $2^{60}/4$ then $c = 12$. In this case, round complexity is $c \log n \cdot c \log n$, i.e., $12 \cdot 60 \cdot 12 \cdot 60 = 518400$ and the experimental rounds are 129600 (see in Figure 3-1). This show that in the experimental set-up, the number of rounds improved (lower) by a factor of 4 when fractions of Byzantine nodes are $1/4$.

Summary of the message complexity: Figure 3-2 shows, the number of message increase for the fixed number of nodes as the number of Byzantine nodes increase. This happens due to the increment of the constant value c with the number of Byzantine nodes. Further, message complexity varies by the number of rounds that send the message during the execution of the algorithm. The

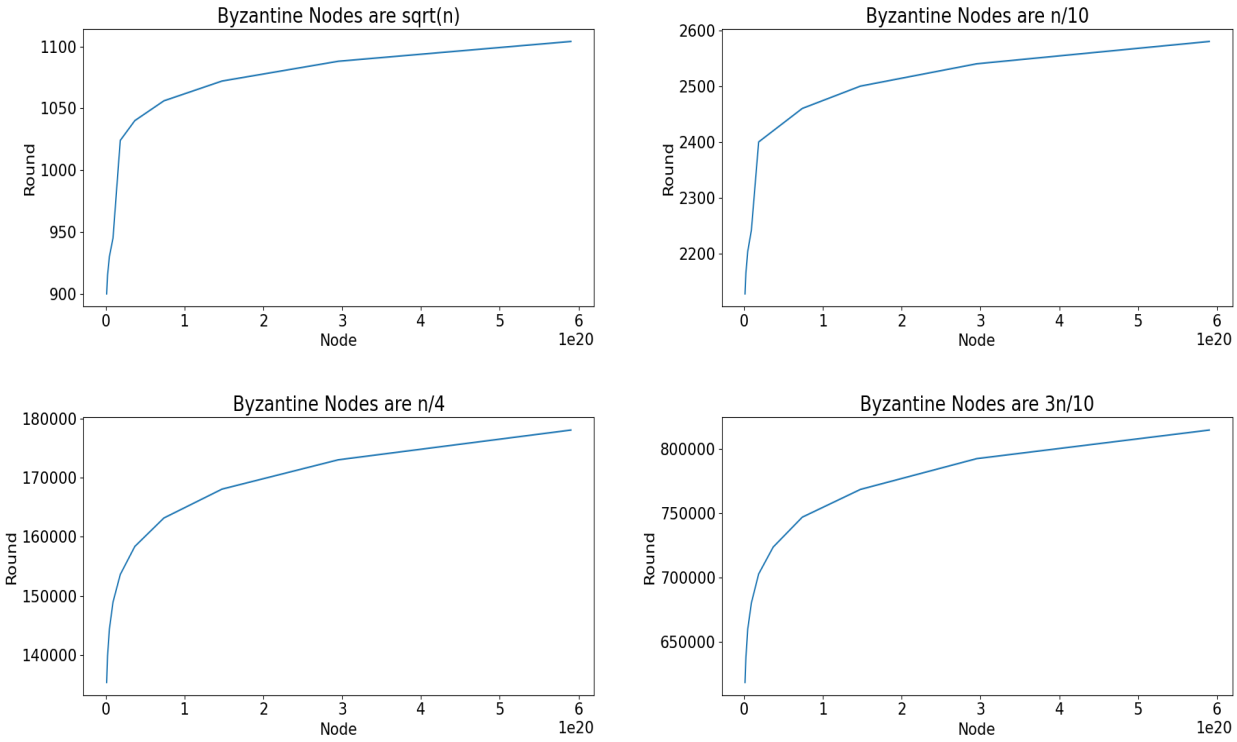


Figure 3-1: Algorithm 3 evaluate the performance w.r.t the different number of Byzantine nodes. X-axis shows the number of nodes in the network and Y-axis shows the number of rounds taken to execute the Algorithm 3.

number of rounds in which honest nodes send the message decreased to constant since Byzantine nodes randomly decide whether to send or not to send the message to the honest nodes. Therefore, honest nodes receive the message from the Byzantine nodes at an early stage of the algorithm. For that reason, the number of messages are lower by a factor of $c \log n$ as compared to the theoretical guarantees. For example, if we consider the case that we considered for the round analysis, in which, we considered the number of nodes, $n = 2^{60}$ and Byzantine nodes $n/4$ then $c = 12$. In that case, theoretical messages are $2 \cdot \sqrt{n \log n} \cdot c^3 \cdot \log^3 n = 6.2 \cdot 10^{18}$. Experimentally, the number of rounds in which messages were sent is 3. Therefore, the experiment takes $0.05 \cdot 10^{18}$ messages (see in the Figure 3-2). This suggests the number of messages sent in the experimental setup is improved (lower) by a factor of 120.

The experiments show that for a wide range of parameters, the algorithm results corroborate our theoretical guarantees. Furthermore, the experimental results perform better than the theoretical guarantees when the Byzantine nodes behave randomly.

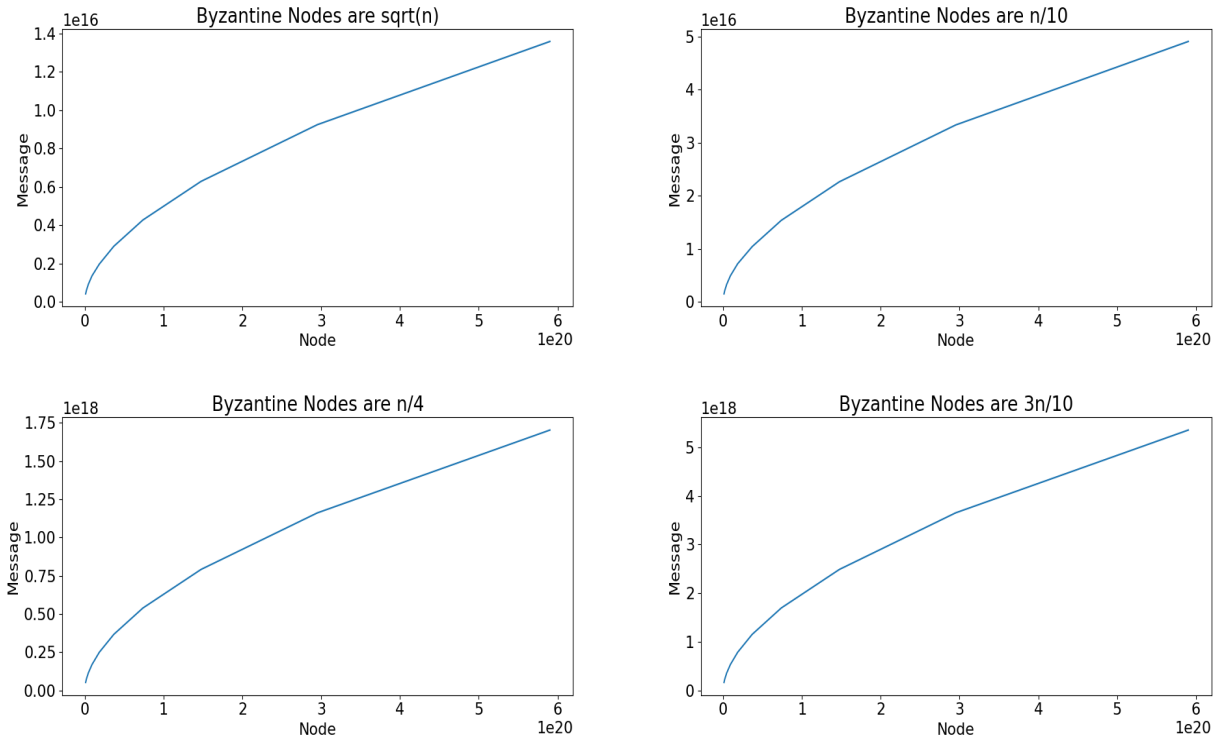


Figure 3-2: Algorithm 3 evaluate the performance w.r.t the different number of Byzantine nodes. X-axis shows the number of nodes in the network and Y-axis shows the number of messages taken to execute the Algorithm 3.

3.7 Conclusion

We studied one of the fundamental problem in distributed networks, namely Byzantine agreement. We showed that implicit Byzantine agreement can be solved with sublinear message complexity in the honest majority setting with the help of cryptographic set up of PKI and hash function, and access to a global coin. The bound is also optimal up to a $\mathrm{polylog}\, n$ factor. To the best of our knowledge, this is the first sublinear message bound result on Byzantine agreement. We also implemented our algorithm to show its efficiency w.r.t different sizes of the Byzantine nodes. We further analyzed some relevant results which immediately follow from our main result. We also studied subset agreement, a generalization of the implicit agreement.

A couple of interesting open problems raised by the work are: (i) is it possible to achieve a sublinear message complexity Byzantine agreement algorithm without the global coin or hash function in this setting? (ii) whether a sublinear message bound is possible under adaptive adversary, which can take over the Byzantine nodes at any time during the execution of the algorithm?

Chapter 4

Tight Bounds on the Fault-Tolerant Graph Realizations in the Congested Clique

In this chapter, we study the graph realization problem in the Congested Clique model of distributed computing under crash faults[1]. We consider *degree-sequence realization*, in which each node v is associated with a degree value $d(v)$, and the resulting degree sequence is realizable if it is possible to construct an overlay network with the given degrees. Our main result is a $O(f)$-round deterministic algorithm for the degree-sequence realization problem in a n-node Congested Clique, of which f nodes could be faulty ($f < n$). The algorithm uses $O(n^2)$ messages. We complement the result with lower bounds to show that the algorithm is tight w.r.t the number of rounds and the messages simultaneously. We also extend our result to the Node Capacitated Clique (NCC) model, where each node is restricted to sending and receiving at-most $O(\log n)$ messages per round. In the NCC model, our algorithm solves degree-sequence realization in $O(nf/\log n)$ rounds and $O(n^2)$ messages. These algorithms work for KT_1 (Knowledge Till 1 hop) model where nodes know their neighbors' IDs.

We further study the graph realization problem in the more robust model that is KT_0 (Knowledge Till 0 hop) model, in which each node knows the IDs of all the nodes in the clique, but does not know which port is connecting to which node-ID. We extend the result to KT_0 when the network is anonymous, i.e., the IDs of the neighboring nodes are unknown. We present an algorithm that solves the graph realization problem in the KT_0 model with matching performance guarantees as in the KT_1 model.

4.1 Introduction

Graph Realization problems have been studied extensively in the literature, mainly in the sequential setting. In general, graph realization problems deal with constructing graphs that satisfy certain

[1]These findings are based on joint work with Anisur Rahaman Molla and Sumathi Sivasubramaniam (which appeared in International Symposium on Algorithmics of Wireless Networks 2022) and contains material from [94]. Further, the findings are based on the work which appeared at International Conference on Distributed Computing and Intelligent Technology 2023 and contains material from [91].

predefined properties (such as a degree sequence). While the area was mostly focused on realizing graphs with specified degrees [68], other properties, such as connectivity [52, 50, 51], flow [62] and eccentricities [23, 102] have also been studied.

The *degree-sequence* realization problem has been explored widely in the centralized setting. Typically, the problem consists of the following. Given a sequence of non-negative numbers $\mathcal{D} = (d_1, d_2, \ldots, d_n)$, the degree-sequence problem asks if $\mathcal{D}$ is *realizable*. A sequence $\mathcal{D}$ is said to be *realizable* if there is a graph of n nodes whose sequence of degrees matches $\mathcal{D}$. The first complete characterization of the problem was established in 1960, when Erdös and Gallai [43] showed that $\mathcal{D}$ is realizable if and only if $\sum_{i=1}^{k} d_i \leqslant k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$ for every $k \in [1, n]$ following which, the definitive solution was independently found by Havel and Hakimi [68, 67], a recursive sequential algorithm that takes $O(n)$ time (see Section 4.3 for more detail). Non-centralized versions of realizing degree sequences have also been studied, albeit to a lesser extent. Arikati and Maheshwari [9] provide an efficient technique to realize degree sequences in the PRAM model.

Recently, Augustine et al. [12] studied the graph realization problem in distributed networks. They approach the graph realization from the perspective of Peer-to-Peer (P2P) overlay construction. P2P overlay networks are virtual networks built on top of the underlying network, e.g., Internet. Given the increasing popularity of P2P networks (in key areas like cryptocurrencies, blockchain, etc.), research on P2P networks, particularly in the construction of P2P overlays, has become a crucial part of distributed computing.

In [12], the authors build overlay networks by adapting graph realization algorithms. Briefly, given a network of n nodes $V = \{v_1, \ldots, v_n\}$, where each v_i is assigned a degree d_i, the goal is to create an overlay graph $G(V, E)$ such that $d(v_i) = d_i$ in G, and for any edge $e \in E$, at least one of e's endpoints knows the existence of e (referred to as an *implicit realization*). Note that, for any edge it is only required that one endpoint must be aware of its existence, which means that a node may be aware of only a small part of the realized graph. In the best case, the nodes know only their neighbors in the final realization. However, in most of the P2P overlay applications, it is crucial to know the entire overlay graph. Furthermore, the network is assumed to be fault-free.

In this work, we extend the graph realization problem in a faulty setting where node may fail by crashing. In addition, we consider the graph realization from the approach of learning degrees so that the nodes may recreate the degree sequence locally, i.e., they know the entire overlay graph. To be precise, we develop fault-tolerant algorithm which ensures that each node learns the degree of all the non-crashed nodes (at least) in the network. This allows the nodes to use the sequential Havel-Hakimi algorithm [68, 67] to build the overlay graph G locally. Note that, since every (non-

crashed) node learns the degree sequence and creates the realization, all of them know the entire overlay graph.

Our work is primarily in the Congested Clique (CC) model, a well-studied model in distributed networks [104]. The Congested Clique model has been explored widely for many fundamental distributed problems [29, 57, 56, 70, 71, 87, 22, 42, 82, 21]. Distributed networks are faulty by nature; nodes crash, links fail, nodes may behave maliciously, etc. In the node failure settings, the two fundamental problems, namely, agreement and leader election has been studied extensively [41, 53, 72, 14, 25, 46, 78, 1, 92, 59]. The main challenge in a faulty setting is to ensure that no two nodes have a different view of a value, which may lead to an increase in the number of rounds or messages or both. Thus, in our work, we focus on developing an algorithm that minimizes the time complexity and the message complexity simultaneously in the Congested Clique with faulty nodes.

In order to make our algorithm more suited for the networks with message overhead limitation (such as peer-to-peer networks), we also extend the algorithm to the Node Capacitated Clique (NCC) model, introduced by Augustine et al. [13]. In the NCC, nodes are restricted to send and receive only $O(\log n)$ messages per round, making it ideal for more realistic settings (e.g., less or no overhead at nodes). The authors in [12] restrict their graph realization algorithms primarily to the NCC model. While our main algorithm considers the CC model, we extend it to the NCC model also. For both the settings (CC and NCC), our algorithms do not require any prior knowledge of f— the number of faulty nodes.

As a byproduct, our algorithms solve fault-tolerant reliable broadcast or consensus [122, 33, 36, 35] in a congested clique. Reliable broadcast algorithms ensure that both senders and receivers agree on a value m sent by a correct process. That is, either all correct processes decide on the value m being sent or decide the sender is faulty. While this is similar to our problem, in our case we extend the basic premise to ensure that all nodes agree on the same degree sequence, which consists of ensuring that the network agrees on at most $O(n)$ values (degrees). This causes a significant challenge to the broadcast protocols in a faulty setting. To the best of our knowledge, the best known for consensus protocol in the crash failure setting has $O(f)$ time complexity and $O(n)$ message complexity [35]. Since there could be $O(n)$ values to be agreed to make sure that all the non-crashed nodes have the same degree sequence, the best-known reliable broadcast or consensus protocol would not give better bounds if applies to solve the graph realization problem in the congested clique.

Chapter Organization: The rest of the chapter is organized as follows. In the rest of the section, we first present our results, followed by a brief description of the model and a formal defini-

tion of our problem. In Section 4.2, we discuss various related works in the field. Section 4.3 provides a brief description of the sequential Havel-Hakimi solution for the graph realization problem. Section 4.4 presents the main result, which is an efficient solution for the fault-tolerant graph realization problem and also provides matching lower bounds. In Section 4.5, we extend the graph realization algorithm to the NCC model. Further, we improve the graph realization results with KT_0 model in Section 4.6. And finally, in Section 4.7 we conclude the chapter.

4.1.1 Our Contributions

We present an efficient algorithm and matching lower bounds for the distributed graph realization problem in the Congested Clique and Node Capacitated Clique model in the presence of node failures. Specifically, our results are:

1. Findings in KT_1 model:

   (a) An $O(f)$-round deterministic algorithm for the graph realization problem in an n-node Congested Clique with at most $f < n$ node failures. The message complexity of the algorithm is $O(n^2)$, where the size of each message is $O(\log n)$ bits.

   (b) We show a matching lower bound for both the time and the message bounds– which demonstrates the simultaneous time and message bounds optimality of our algorithm.

   (c) We extend the algorithm for the Congested Clique to the Capacitated Clique, and present a $O(nf/\log n)$-round and $O(n^2)$-message complexity algorithm.

2. Findings in KT_0 model:

   (a) Consider an n-node Congested Clique with KT_0 model, in which $f < n$ nodes may crash arbitrarily at any time. Given an arbitrary n-length degree sequence $\mathcal{D} = (d_1, d_2, \ldots, d_n)$ as an input such that each d_i is only known to one node in the clique. Then there exists a deterministic algorithm that solves the fault-tolerant graph realization problem in $O(f)$ rounds and using $O(n^2)$ messages such that f is unknown to the network.

   (b) We show a matching lower bound for both the time and the message bounds– which demonstrates the simultaneous time and message bounds optimality of our algorithm.

4.1.2 Model and Definitions

We consider the message-passing model of distributed computing. The underlying network is a Congested Clique (CC) [87, 104]. A Congested Clique consists of n nodes which are identified by unique IDs from $\{u_1, u_2, \ldots, u_n\}$. Communication among the nodes happens in synchronous rounds, where in each round a node can communicate with any of the other $n-1$ neighbors by sending a message of size $O(\log n)$ bits. This was first introduced in [118]. Nodes know the ID of the other nodes and the port connecting to the IDs. This assumption is known as the KT_1 model [118]. On the other hand, in KT_0 model (knowledge till 0 hop), a node does not know the IDs of its neighbors initially.

We assume that an arbitrary subset of the nodes in the clique of size up to $f < n$ may fail by crashing. A faulty node may crash in any round during the execution of the algorithm. If a node crashes in a round, then an arbitrary subset (possibly all) of its messages for that round may be lost (as determined by an adversary) and may not have reached the destination (i.e., neighbors). The crashed node halts (inactive) in the further rounds of the algorithm. If a node does not crash in a round, then all the messages that it sends in that round are delivered to the destination.

We consider an *adaptive* adversary controls the faulty nodes, which selects the faulty nodes at any time during the execution of the algorithm. Also, the adversary chooses when and how a node crashes, which is unknown to the nodes.

The time complexity of an algorithm is the number of rounds from the start until the termination of the algorithm. The message complexity of an algorithm is the total number of messages sent by all the nodes throughout the execution of the algorithm.

In the interest of being useful in the Peer-to-Peer context, we also consider the Node Capacitated Clique (NCC) [13] model. NCC limits each node to send or receive a bounded number of messages, which, interestingly, makes NCC quite distinct from CC. In this model, any node, say, u can send or receive $O(\log n)$ messages, each of size $O(\log n)$ bits.

We will now formally define the distributed graph realization problem (with and without faults). We say that an overlay graph $G = (V, E)$ is constructed if, for every $e = (u, v) \in E$, at least one of the endpoints is aware of the ID of the other and also aware that $e \in E$. We say that the overlay graph is *explicit* if, for every edge $e \in E$ in the graph both endpoints know each other's ID and are aware of that $e \in E$.

Definition 4.1 (Distributed Graph Realization [12])**.** *Let $V = \{v_1, \ldots, v_n\}$ be the set of nodes in the network. Let $\mathcal{D} = (d_1, d_2, \ldots, d_n)$ be an input degree sequence such that each d_i is only known*

to the corresponding node v_i. The distributed degree realization problem requires that the nodes in V construct a graph realization of $\mathcal{D}$ such that in the resulting overlay graph G, the following conditions hold:

1. *The degree sequence of G is precisely $\mathcal{D}$.*
2. *The degree of v_i is d_i, $\forall i \in \{1, \ldots, n\}$.*

Thus, in the case of the distributed graph realization problem, a solution should output the graph G if $\mathcal{D}$ is a realizable degree sequence; otherwise, the output is "unrealizable".

Definition 4.2 (Distributed Graph Realization with Faults)**.** *Let $V = \{v_1, \ldots v_n\}$ be the set of nodes in the network and $\mathcal{D} = (d_1, d_2, \ldots, d_n)$ be an input degree sequence such that each d_i is only known to the corresponding node v_i. Let $F \subset V$ be an arbitrary subset of faulty nodes in the network, such that $|F| = f \leq n - 1$. Let us define $\mathcal{D}' \subseteq \mathcal{D}$ be the modified degree sequence after losing the degrees of some faulty nodes and G' be the corresponding overlay graph over $\mathcal{D}'$. The distributed degree realization with faults problem requires that the* non-faulty *nodes in V construct a graph realization of $\mathcal{D}'$ such that in the resulting overlay graph G', the following conditions hold:*

1. *$|\mathcal{D}'| \geq n - |F|$.*
2. *$\mathcal{D} - \mathcal{D}'$ is the degree sequence corresponding to some nodes that crashed.*
3. *For any edge $e = (u, v) \in G'$, either u or v (or both) knows of the existence of e.*

The required output is an overlay graph G' if $\mathcal{D}'$ is realizable; otherwise, the output is "unrealizable".

4.2 Related Work

Fault-tolerant computation has always been a popular area of research in distributed computation, only becoming more popular with the prevalence of P2P networks that encourage for high decentralization. Often the focus of such research is on maintaining connectivity [16], recovery, or on ensuring that the network can tolerate a certain number of faults [133]. We refer interested readers to [127] for more information on models and techniques for designing such systems. In our work, we focus mainly on ensuring that all the (non-faulty) nodes have the same view of the information in-spite of the presence of numerous faults.

In lieu of this, our goal in this work is to solve the distributed degree sequence problem, which is a graph realization problem. Graph realization problems have been well studied in the literature, focusing on problems such as realizing graphs with specified degrees [68] as well as other properties, like connectivity and flow [62, 52, 50, 51] or eccentricities [23, 102]. Bar-Noy et al. explored the graph realizations in various settings like vertex-weighted, distance set, and relaxed and approximate graph realizations [19, 18, 20]. Arikati and Maheshwari [9] provide an efficient technique to realize degree sequences in the PRAM model, and in [12], the authors explored graph realization from a distributed setting. However, both of these works assumed a fault-free setting. Here we extend the model to a faulty setting. In [12], Augustine et al. discussed distributed graph realizations on a path (both implicit and explicit). However, in their work, a node is only required to learn its neighbors in the final realization, in our work, the nodes are aware of the entire graph.

In the area of P2P overlays, there is a great deal of existing literature. In particular, a large amount of research exists to create overlays that provide structure and stability. This is best captured by overlays such as Chord [131], CAN [121], and Skip Graphs [10]. Overlays have also been specifically designed to tolerate faults [47, 15]. For a more detailed survey on P2P overlays and their properties, we refer interested readers to the following excellent surveys [107, 105].

Our network is modeled using the Congested Clique model. The congested clique model, first introduced by Lokter et al., [104] has been well studied, in both the faulty and non-faulty settings [115, 38]. Problems such as agreement and leader election have also been well studied in this model [1, 14, 59, 90]. To the best of our knowledge, this is the first time graph realization problems have been studied in the faulty setting of Congested Clique.

While the results for the Congested Clique are interested in and of itself, in order to make our work more applicable to P2P settings, we also explore how to solve the graph realization problem in the NCC [13]. In the NCC, unlike the CC, a node is allowed to send/receive at most $O(\log n)$ messages in a round, this makes gathering information in a faulty-setting slightly more challenging.

4.3 Preliminary: The Sequential Havel-Hakimi Algorithm for Graph Realization

We will now briefly introduce the sequential Graph Realization problems that inspired our distributed version of the same. Graph realization problems are fairly simple in characterization. The basic premise is as follows: given a particular degree sequence $\mathcal{D} = (d_1, d_2 \dots d_n)$, can we create a graph G whose degree sequence is precisely $\mathcal{D}$? The most well known characterization is given

independently by Havel [68] and Hakimi [67], which can be stated concisely as follows.

Theorem 4.1 (Based on [68] and [67]). *A non-increasing sequence $\mathcal{D} = (d_1, d_2, ..., d_n)$ is graphic (i.e., graph is realizable) if and only if the sequence $\mathcal{D}' = (d'_2, ..., d'_n)$ is graphic, where $d'_j = d_j - 1$, for $j \in [2, d_1 + 1]$, and $d'_j = d_j$, for $j \in [d_1 + 2, n]$*

This characterization directly implies a $O\big(\sum_{i=1}^{n} d_i\big)$ time (in terms of number of edges) sequential algorithm, known as the Havel-Hakimi algorithm, for constructing a realizing graph $G = (V, E)$ where $V = \{v_1, ...v_n\}$ and $d(v_i) = d_i$, or deciding that no such graph exists. The algorithm works as follows. Initialize $G = (V, E)$ to be an empty graph on V. For $i = 1$ to n, in step i do the following:

1. Sort the remaining degree sequence in non-increasing order ($d_i \geq d_{i-1} \geq \ldots d_n$).
2. Remove d_i from $\mathcal{D}$, and set $d_j = d_j - 1$ for all $j \in [i + 1, d_i + i + 1]$.
3. Set the neighborhood of the node v_i to be $\{v_{i+1}, v_{i+2}, ...v_{i+1+d_i}\}$.

If, at any step, $\mathcal{D}$ contains a negative entry, the sequence is not realizable.

4.4 Fault-Tolerant Graph Realization in KT_1

In this section, we present an efficient solution for graph realization with faults. Recall that we are given a n-node Congested Clique, in which at most $f < n$ nodes may crash arbitrarily at any time. Also, a vector of degree sequence $(d_1, d_2, \ldots, d_n)$ is given as an input such that each d_i is only known to one node in the clique. Our goal is to construct an overlay graph of size at least $n - f$ if realizable; otherwise output is unrealizable. We present an algorithm that guarantees, despite $f < n$ faulty nodes, that (i) all the (non-crashed) nodes learn and recreate a degree sequence whose size is at least $n - f$, and (ii) this degree sequence is the same for all the nodes as a requirement of the graph realization (see Definition 4.2). This allows the non-crashed nodes to locally realize the overlay graph with the help of Havel-and-Hakimi's algorithm, described in Section 4.3. We note that the algorithm does not require any knowledge of f. Our algorithm crucially uses only a few number of nodes to be involved in propagating the information about crashed nodes to the other nodes in the network – which helps to minimize the message complexity and round complexity of the algorithm.

At a high level, the algorithm runs in two phases. In the first phase, which consists of only two rounds, each node sends the message containing its ID and input-degree twice (in two consecutive rounds). Based on the frequency of the received message, i.e., zero or one or two times, the receiver node considers the sender node's status as faulty or non-faulty. Then each node locally creates an initial faulty list and a degree sequence (known only to itself). In the second phase, through sharing the information about faulty nodes present in the faulty list, the nodes rectify the degree sequence and create a final degree sequence $\mathcal{D}'$, which is guaranteed to be the same for all the non-crashed nodes. The non-crashed nodes then realize the overlay graph using $\mathcal{D}'$ as the degree sequence via the Havel-and-Hakimi algorithm.

Let us recall a few basic assumptions. For simplicity, we assume the KT_1 version of the Congested Clique model, in which all the nodes know the IDs of the nodes in the clique and the corresponding link or port connecting to the IDs. Thus, in the KT_1 version, a node can sort all the nodes in the clique according to their distinct IDs. W.l.o.g, let us assume the IDs are $U = \{u_1, \ldots u_n\}$ in the sorted order, i.e., $u_i < u_j$ for $i < j$. A node knows its position in the sorted order. Notice that a node u_i can track whether it has received any messages from a node u_j. However, the algorithm also works in the KT_0 version of the Congested Clique model where only the IDs are known, but the corresponding links are unknown (i.e., a node does not know which neighbor has which ID).

We will now explain the algorithm in detail, whose terminologies and definitions are summarized in Table 4.1. A complete pseudocode is given in Algorithm 4. The first phase (which we call the *initialization phase*) consists of two rounds. In both rounds, every node broadcasts its input degree value to reach all the nodes in the clique. After the two rounds, each node u_i creates a *faulty-list* F_{u_i} and a *degree sequence* $\mathcal{D}'_{u_i}$ as follows. F_{u_i} consists of n cells corresponding to all the nodes $u_j \in U$ (in the sorted order). For any u_j $(j \neq i)$, if u_i hears from u_j in both the rounds, u_i includes u_j's degree value $d(u_j)$ in its final degree sequence $\mathcal{D}'_{u_i}$ (and correspondingly u_j's entry in the faulty-list is empty, i.e., $F_{u_i}(u_j) = null$). If u_i hears from u_j only in the first round (i.e., u_j crashed during the initialization phase), then u_j is included as a *faulty-node* in its corresponding entry. If u_i did not hear from u_j in both the rounds (i.e., u_j crashed in the first round itself), then u_j is additionally marked as a *smite-node* in u_i's *faulty-list*. At the end of the algorithm, we show that $\mathcal{D}'_{u_i} = \mathcal{D}'_{u_j}$ for all i, j, which represents the final *non-faulty-list* $\mathcal{D}'$.

In the second phase, nodes update their faulty-list locally. Nodes are divided into two groups based on the two states – *active* state and *listening* state. A node in the active state transmits messages, whereas a node in the listening state only receives messages. Nodes may update their faulty-list according to the information received from the active nodes. Initially, all the nodes start

in the listening state. A node u_i in listening state continues to be in that state as long as there is another active node in the network. Let u_j be the ID of the last active node (if there is none, let $j = 1$). A node u_i changes its status from listening state to active state if and only if u_i has not received any message from the last $3(i-j)$ rounds where j is the index of last heard node, i.e., u_j. Then u_i becomes active in the $(3(i-j)+r)^{th}$ round; r is the round when u_i heard from u_j.

After the initialization phase (i.e., from the 3^{rd} round), the node with the minimum ID (or minimum index) becomes the active node, say the node u_1 (if present). If F_{u_1} is non-empty, then u_1 sends the $\langle ID, d(ID)\rangle$ from the first non-empty cell in F_{u_1} twice (in two consecutive rounds). If there is no $d(ID)$ then u_1 sends only ID. This situation may arise when an ID crashed such that u_1 does not receive $d(ID)$ in the initialization phase (in the first two rounds). u_1 continues sending $\langle ID, d(ID)\rangle$ from its F_{u_1} one by one from minimum ID to maximum ID and for each message $\langle ID, d(ID)\rangle$ it sends twice in consecutive rounds. When u_1 finishes sending all the non-null entries in F_{u_1}, it switches to the exit state (which we will explain shortly). If at any point in time, u_1 crashed then u_2 will become the next active node after $3(2-1) = 3$ rounds. The exit state for any node u_j consists of two tasks, (i) move all the F_{u_j} (if any) into $\mathcal{D}'_{u_j}$. (ii) Send out "all-okay" message to all other nodes and exit the algorithm. Sending or receiving "all-okay" message conveys that all the known faulty-nodes have been addressed. All other nodes who hear the message (those who are in the listening state) "all-okay" also enter the exit state. Let us now discuss the update process of nodes in active and listening states in detail. Suppose u_i is a node which is currently in its active state and F_{u_i} is non-empty. Then u_i sends the first non-null entry, say s, twice in consecutive rounds. If s was a *smite-node* then u_i permanently removes s from the F_{u_i}. If s was a *faulty-node*, then u_i moves $d(s)$ to $\mathcal{D}'_{u_i}$. If F_{u_i} is empty, then u_i enters the exit state.

If instead the node u_j happened to be in its listening state, then let i be the index of the last node heard by the node u_j. If u_j has not heard from any node during the last $3(j-i)$ rounds, then u_j sets its state to active. Since each node needs to send the message twice, we maintain a gap of one extra round (total three) to prevent two or more nodes from being active at the same time. But if u_j is currently receiving messages from an active node u_i then it does the following. Let s be the ID it heard from u_i. u_j now updates its faulty-list F_{u_j} based on how many times u_j heard about s from the active node u_i.

If u_j heard about s twice, and s was a *smite-node* then u_j permanently removes s from its faulty-list F_{u_j}. However, if s was a *faulty-node* then u_j moves s to $\mathcal{D}'_{u_j}$ (if not in $\mathcal{D}'_{u_j}$) and permanently removes $F_{u_j}(s)$ (if any). Notice that u_j might have received s earlier two times, in that case $F_{u_j}(s)$ will be null and there will be a corresponding entry in the $\mathcal{D}'_{u_j}$. If u_j heard about s only once, then if s had been a *smite-node* (or respectively a *faulty-node*) then $F_{u_j}(s)$ is marked as a *smite-*

node (respectively *faulty-node*). All non-null entries in F_{u_j} below s's index are included in $\mathcal{D}'_{u_j}$ if the entries correspond to faulty-nodes, otherwise they are set to null.

Throughout the algorithm, inclusion of the degree in $\mathcal{D}'$ is permanent. Therefore, the corresponding node entries in *faulty-list* remain null. At last, all the non-faulty nodes have the same view of *non-faulty-list* i.e, $\mathcal{D}'$. Therefore, the graph can be realized locally by the Havel-Hakimi's algorithm (see Section 4.3).

We will now show the correctness of the algorithm using the following lemmas and then analyze the time and the message complexity. Lemma 4.1, Lemma 4.2 and Lemma 4.3 show that the final degree sequence $\mathcal{D}'$ of all the non-crashed nodes are the same. Thus, the algorithm correctly solves the distributed graph realization with faults in the Congested Clique. Finally, Lemma 4.4 analyzes the time complexity and Lemma 4.5 analyzes the message complexity of the algorithm.

Algorithm 4 FAULT-TOLERANT-GRAPH-REALIZATION

Require: A Congested Clique of n nodes $U = \{u_1, u_2, \ldots, u_n\}$. Each node u_i is given a degree value $d(u_i)$ as an input.

Ensure: A corresponding graph realization that satisfies the conditions for distributed graph realization with faults.

1: For the first two rounds, each node u_i broadcasts $\langle u_i, d(u_i)\rangle$ to all the nodes. Each node creates the faulty-list F_{u_i} and the degree sequence $\mathcal{D}'_{u_i}$.
2: Nodes are either in active, listening, or exit state. If u_1 is non-faulty, then u_1 becomes the first active node.
Nodes in Active State.
3: **while** F_{u_i} is non empty **do**
4: u_i sends the $\langle ID, d(ID)\rangle$ (send $d(ID)$ degree is available otherwise indicate smite if it is a smite node) from minimum ID to maximum ID in F_{u_i} twice – in two consecutive rounds.
5: **if** sent ID is a *smite-node* **then**
6: Remove ID from the list F_{u_i}.
7: **else if** sent ID is *faulty-node* **then**
8: Move the degree with ID to $\mathcal{D}'_{u_i}$.
9: **end if**
10: **end while**
11: Switch to exit state.
Nodes in Listening State follows the Algorithm 5
Nodes in Exit State
12: Remaining faulty nodes in the faulty-list are moved to $\mathcal{D}'$.
13: Send out an *"all-okay"* message to all and exit the algorithm.
14: All the non-crashed nodes have the same view of $\mathcal{D}'$. Therefore, graph can be realized by all the non-crashed nodes using the Havel-Hakimi's algorithm.

Algorithm 5 NODES IN LISTENING STATE FOR THE ALGORITHM 4

if nodes receives the message *"all-okay"* from any node **then**
    Switch to exit state.
end if
For any listening node u_j, let i be the index of the last node u_i heard by the node u_j. (starts with $i = 1$).
if u_j has not heard from any node in the last $3(j - i)$ rounds. **then**
    u_j switches to active state.
end if
if Node u_j heard from a node u_i **then**
Let s be the ID heard from u_i. u_j updates its list according to whether it heard s twice or once.
Heard Twice:
    if s is a *smite-node* **then**
        Remove s from the faulty-list permanently. All non-null entries before s's index are moved appropriately.
    end if
    if s is a faulty-node **then**
        Move s's degree to $\mathcal{D}'_{u_j}$. All non-null entries below s's index are moved appropriately.
    end if
end if
Heard Once:
if s is a smite-node **then**
    Set $F_{u_j}(s)$ as smite node.
end if
if s is a faulty-node **then**
    Set $F_{u_j}(s)$ as faulty-node.
end if

The *view* of a node s at u is the classification of s as a *smite-node* or *faulty-node* at u. We will now show that for any node s, the view of s is same across all the nodes at the end of the protocol.

Lemma 4.1. *Let s be an ID for which there are conflicting views at the beginning of the second phase. Then at the end of the protocol, i.e., when all nodes have reached the exit state, all nodes are guaranteed to have the same view of s.*

Proof. We prove this in cases. Suppose s is sent by some node. We look at the possible scenarios for both the heard-twice and the heard-once.

Heard Twice: For the trivial case when a good node successfully broadcasts s twice to all nodes, the statement follows immediately. For the other case when a faulty node successfully broadcasts s in the first round, but crashes in the second round, there is a possibility that some nodes may have

heard about s twice, but the others may have not. In this case, all the nodes are still guaranteed to have heard s at least once (in the first round of transmission) and thus would have the same view, as they would have all updated their views simultaneously.

Heard Once: Let us consider the two cases.
Case 1: Node crashed sending s during the second round. This scenario is equivalent to the second scenario in the heard-twice case, and thus follows the same logic.
Case 2: Node crashed sending s during the first round. In this case, let u_i and u_j be two nodes that survived until the end, then there must exist a round in which u_i (or respectively u_j) informed each other about their view of s. Then, based on the received values, they updated their views to match. □

Lemma 4.2. *Let s be an ID that was successfully transmitted twice by an active node u_i during the second phase of the algorithm. Then for any pair of nodes u_j, u_k $(j, k \geq i)$, all the (non-null) entries below s are the same in their faulty-list. That is, for any non-null entry, $F_{u_j}(p) = F_{u_k}(q)$ for all $p, q < s$.*

Proof. Suppose not. Let there be a node r ($r < s$) such that for two nodes u_j, u_k, $F_{u_j}(r)$ reads as *smite-node* while $F_{u_k}(r)$ reads as *faulty-node*. If r had crashed during the second round of phase 1, this immediately contradicts our claim, as all nodes can only have r as either $null$ or *faulty-node*. In case it crashed during the first round, consider the following. Since r's cell in u_i is null (otherwise $r's$ value would have been transmitted first by u_i), u_i must have heard r successfully twice at a previous round of either phase. Which implies that all other nodes would have heard about r at least once, hence would share the same view of r. □

Lemma 4.3. *If a node u_i decides to exclude an entry from its faulty-list then all other nodes u_j $(j \neq i)$ will also eventually exclude that entry. Conversely, if u_i moves an entry from its fault-list to $\mathcal{D}_{u_i}$, then eventually all other nodes u_j will also move that entry to their $\mathcal{D}_{u_j}$.*

Proof. If a node s is excluded from u_i's *faulty-list* then u_i must have heard s as a *smite-node* twice, which indicates that all other nodes must have received s at least once (as a *smite-node*). So all the nodes exclude s from their faulty-list. The same logic applies in case of moving an entry to $\mathcal{D}_{u_i}$. □

Lemma 4.4. *The time complexity of the Algorithm 4 is $O(f)$ rounds.*

Proof. The first phase takes exactly two rounds. For the finalization of $\mathcal{D}'$, any non-faulty node would require at most f messages in $O(f)$ rounds. In case of faults, there can be at most f node-crashes, which can introduce a delay of at most f rounds. During phase 2, a node may take at most

$O(f)$ rounds to inform the network of the entries in its fault-list (including delay due to faults). Therefore, round complexity is $O(f)$. □

Lemma 4.5. *The message complexity of the Algorithm 4 is $O(n^2)$, where each message is at most $O(\log n)$ bits in size.*

Proof. The first phase of the algorithm uses $O(n^2)$ messages, since all the nodes broadcast in two rounds. In the second phase, each faulty node's ID is broadcast only twice by a single (non-faulty) node. Since there are at most f faulty nodes, $O(nf)$ messages incur. Therefore, total message complexity is $O(n^2)$, since $f < n$. Since both the value of degree and the ID can be encapsulated using $O(\log n)$ bits, messages are at most $O(\log n)$ bits in length. □

Thus, we get the following main result of fault-tolerant graph realization.

Theorem 4.2. *Consider an n-node Congested Clique, where $f < n$ nodes may crash arbitrarily at any time. Given an n-length graphic sequence $\mathcal{D} = (d_1, d_2, \ldots, d_n)$ as an input such that each d_i is only known to one node in the clique, there exists an algorithm which solves the fault-tolerant graph realization problem in $O(f)$ rounds and using $O(n^2)$ messages.*

Remark: 2 (Extension to KT_0 Model)**.** *The same message and round complexity is achievable in the KT_0 model, in which each node knows the IDs of all the nodes in the clique, but does not know which port is connecting to which node-ID. Algorithm 4 can be easily modified to work in this KT_0 model. During the first two rounds (i.e., in the initialization phase), each node keeps track of the links or ports through which it receives the messages. If it receives a message $\langle u_j, d(u_j)\rangle$ through some port, that port must be connecting to node-ID u_j. At the same time, a node can identify the faulty nodes' ID from which it did not receive any messages. If not received in the first round itself, the ID is considered as a smite node. Once we have the initial faulty-list and the degree sequence, then we run the second phase of the Algorithm 4.*

4.4.1 Lower Bound

In the graph realization with faults problem, the nodes are required to learn the degrees assigned to the other nodes in the network and recreate a degree sequence which must be the same for all the nodes. If two nodes have a different view of the final degree sequence $\mathcal{D}'$, then their output would be different. It essentially reduces to a *consensus* problem where all the nodes agree on the degrees in $\mathcal{D}'$. In the consensus problem, all nodes start with some input value and the nodes are required to agree on a common value (among all the input values to the nodes). In the presence of

Terminology at a Glance for a node u	
Terminology	Definition
Smite-node*	Node $v \neq u$ is classified as *smite-node* if u did not receive v's degree in the first two rounds.
Smite-node (Phase 2)	Node $v \neq u$ is classified as *smite-node* if u receives a message from any node (even once) which classifies v as a *smite-node*.
Faulty-node	Node $v \neq u$ is classified as a *faulty-node* if u receives v's degree only once.
Faulty-list (F_u)	List of known faulty IDs (and their corresponding degrees if present) at u.
Listening State	A node is in the listening state if it is waiting for its turn (to send entries from its faulty-list) or to receive an "all-okay" message.
Active State	A node is in the active state if it is transmitting the entries from its faulty-list.
"all-okay"	Reception of "all-okay" at u acts as a signal for u to terminate the algorithm.
Degree Sequence ($\mathcal{D}'_u$)	$\mathcal{D}'_u$ keeps track of all the degrees heard from other nodes in the network. At termination, it contains the final degree sequence used for graph realization at u.

Table 4.1: Terminology and their definition used throughout the algorithms in the Section 4.4 and Section 4.5. * represent the terminology's definition for Phase 1.

faulty nodes, all non-faulty nodes must agree on a common value. Therefore, any t-round solution of the graph realization with faults problem can be used to solve the consensus problem in $O(t)$ rounds. Now, a lower bound of $f + 1$ on the number of rounds required for reaching consensus in the presence of f crash failures is a well-known result (see Chapter 5: Fault-Tolerant Consensus of Attiya-Welch's book [11]). Thus, this $f + 1$ lower bound on the round complexity also applies to the graph realization with faults problem. Hence, we get the following result.

Theorem 4.3. *Any algorithm that solves the distributed graph realization with faults in an n-node Congested Clique with f crash failures requires $\Omega(f)$ rounds in some admissible execution.*

We now argue that graph realization with faults, where nodes construct the entire overlay graph,

requires $\Omega(n^2)$ messages in the Congest model [118]. Consider a graph realization algorithm $\mathcal{A}$ that constructs the entire overlay graph in a n-node Congested Clique. The algorithm $\mathcal{A}$ must correctly output the overlay graph for any inputs.

First consider the fault-free case, i.e., no faulty nodes in the Congested Clique. Since every node constructs the entire overlay graph, the n degree-values (for all the nodes) need to be propagated to all the nodes. The size of a degree can be $O(\log n)$ bits. In the Congest model, the size of each message is $O(\log n)$ bits. Hence, a node can send at most a constant number of degree-values in one message packet. Now, it requires $n-1$ messages to send one degree-value from a node to all the nodes. Thus, it requires at least $n(n-1) = O(n^2)$ messages to propagate n degree-values to all the nodes.

Consider the faulty case. The following situation may occur in some execution of $\mathcal{A}$. A faulty node may crash in a round by sending $O(n)$ messages in that round. Further, any message of a crashed node that has not reached to all the nodes may need to be rectified to make it consistent throughout the network. This requires that the message may need to be propagated to all the nodes. Hence, we require at least $O(n^2)$ messages in the worst case. Thus, we get the following result.

Theorem 4.4. *In the* Congest *model, any graph realization algorithm, in which nodes construct the entire overlay graph, in a n-node network (with or without faults) requires $\Omega(n^2)$ messages in some admissible execution.*

4.5 Fault-Tolerant Graph Realization in the KT_1 NCC Model

In this section, we extend the above fault-tolerant algorithm to the Node Capacitated Clique (a.k.a. NCC) model. To the best of our knowledge, distributed versions of graph realizations problem (without faults) were first studied by the authors in [12]. In the original work, the authors attempted to solve the distributed degree sequence problem in two versions of the NCC model introduced by the authors in [13], namely the NCC_0 and the NCC_1. In this section, we present how we may extend our ideas for solving the graph realizations with faults problem in the Congested Clique to NCC_1.

Briefly, the NCC_1 is exactly like the KT_1 version of the Congested Clique (CC) model with one clear difference, which is a constraint on the number of messages a node is allowed to send/receive in a round. In the NCC_1, a node is allowed to send or receive at most $O(\log n)$ messages in a round, unlike the CC model, in which we do not have a bound on the number of messages. When a node receives more than $O(\log n)$ messages, it chooses to drop the excess. Note that the model imme-

diately implies a $n/\log n$ lower bound in terms of time complexity for our version of the graph realization, as each node needs to learn $n-1$ degrees in a clique. Our solution takes $O(nf/\log n)$ rounds in the NCC_1 model, but it is optimal in the number of messages.

We will now present how we modify Algorithm 4 for the NCC_1. The key idea is to change how each node sends its degree to every other node in the network. We leverage the idea of parallelism and cyclic permutation so that each node can in one round (i) inform $O(\log n)$ other nodes and (ii) receive the degree (or faulty IDs as the case may be) from at most $O(\log n)$ other nodes.

The steps above gives us the following straightforward idea, divide the nodes into $n/\log n$ groups $g_1, g_2 \ldots, g_{\frac{n}{\log n}}$ such that each group has no more than $O(\log n)$ nodes. This allows all nodes in a group g_i to send their degree to all nodes in the group g_j while satisfying the message constraints present in NCC_1.

By taking advantage of the parallelism present in the setting (and working with different permutations in each round) we can guarantee that all nodes learn the degrees in the network in $O(n/\log n)$ rounds. This gives the idea of global broadcast (Algorithm 6).

Algorithm 6 GLOBAL-BROADCAST

1: **for** $r = 0$ to $n/\log n - 1$ **do**
2: **for** $j \in [1, \frac{n}{\log n}]$ in parallel **do**
3: $dest = (r + j) \pmod{\frac{)}{n}\log n}$
4: Nodes in group g_j send their degrees to all the nodes in group g_{dest}.
5: **end for**
6: **end for**

Clearly, the procedure in Algorithm 6 ensures that n nodes may send n messages each over a period of $n/\log n$ rounds. We may use global broadcast to guarantee that up to n non-faulty nodes may send their degrees to the network in $n/\log n$ rounds, but no node receives more than $O(\log n)$ degrees. We use a similar approach when it comes to sending out "all-okay", the difference being we need to ensure that no two nodes send "all-okay" to the same group at once.

Algorithm 7 ensures that all nodes may receive "all-okay" while making sure that two active nodes are not sending the "all-okay" message to the same group of nodes at the same time. We can now modify Algorithm 4 for the NCC. It follows the same steps, the key difference is in the fact that when an active node is sending entries from its faulty-list, it requires $O(n/\log n)$ rounds to inform all the nodes in the network. Since there can be f entries in the worst case scenario, this means that it can take $O(nf/\log n)$ rounds to ensure that all nodes have the same degree sequence.

The process for local update of entries in the faulty-list remains the same as in Algorithm 4

Algorithm 7 GLOBAL-UPDATE

1: **Global Update:** This protocol is executed at u_i when it receives an all-okay message.
2: **for** $j = i$ to $n/\log n$ **do**
3: u_i sends all-okay to all nodes in g_j.
4: **end for**
5: **for** $j = 1$ to $j = i - 1$ **do**
6: u_i sends "all-okay" to all nodes in g_j.
7: **end for**

Now we argue that the Algorithm 8 (below) has the same guarantees for the view of $\mathcal{D}'$ as in Algorithm 4. In fact, we show that the final degree sequence $\mathcal{D}'$ of all the non-crashed nodes are the same (using the similar arguments as in Lemma 4.1, Lemma 4.2, and Lemma 4.3).

Lemma 4.6. *Algorithm 8 guarantees that all nodes have the same view of the degree sequence $\mathcal{D}'$ at the end of the algorithm.*

Proof. Suppose not. Then there exists a node s for which nodes u_i and u_j have different views. If both the nodes survived until the end, there must exist a round where either u_i or u_j would have informed the other of s's status, thus ensuring that they both have the same view of s. Also, the arguments of Lemma 4.2, Lemma 4.3 are still applicable in the NCC case, ensuring that no degree is excluded from some nodes of the network, while being included in the final degree sequence of the other nodes. Hence, the $\mathcal{D}'$, created at the end of the Algorithm 8, is the same across all nodes. □

Note that a key difference in Algorithm 8 to the one designed for the Congested Clique model is that a single loop to inform all nodes of a fault takes $O(n/\log n)$ rounds. Since there are f faults, we have the following theorem:

Theorem 4.5. *Algorithm 8 solves graph realization with faults in the Node Capacitated Clique model in $O(nf/\log n)$ rounds and uses $O(n^2)$ messages.*

4.6 Graph Realization with Faults in KT_0

This section focus on the graph realization problem in the KT_0 model. Recall that we are given an n-node Congested Clique anonymous network, i.e., nodes do not know each other's communication link with respect to their IDs, in which at most $f < n$ nodes may crash arbitrarily at any

Algorithm 8 GRAPH-REALIZATION-WITH-FAULTS-IN-NCC

Require: A Congested Clique of n nodes $U = \{u_1, u_2, \ldots, u_n\}$. Each node u_i is given a degree value $d(u_i)$ as an input.

Ensure: A corresponding graph realization that satisfies the conditions for distributed graph realization with faults.

Using the global broadcast procedure in Algorithm 6, each node u_i broadcasts $\langle u_i, d(u_i)\rangle$ to all the nodes.
Each node creates the faulty-list F_{u_i} and the degree sequence $\mathcal{D}'_{u_i}$.
Nodes are either in active, listening, or exit state. If u_1 is non-faulty, then u_1 becomes the first active node.
Nodes in Active State.
if u_i is an active node **then**
    if F_{u_i} is non empty **then**
        The following for loop is repeated twice.
        for $j \in [1, \frac{n}{\log n}]$ in parallel **do**
            u_i sends the $\langle ID, d(ID)\rangle$ (send $d(ID)$ if available otherwise it is a smite node) from minimum ID to maximum ID in F_{u_i} to g_j. If the list becomes empty, then switch to exit state.
        end for
        if sent ID is a *smite-node* **then**
            Remove ID from its faulty-list.
        else if sent ID is *faulty-node* **then**
            Remove ID from faulty-list. Move ID and ID's degree to $\mathcal{D}'$.
        end if
    else
        Enter exit state.
    end if
end if
Nodes in Listening State
if received the message *all-okay* from any node **then**
    Switch to exit state.
end if
For any listening node u_j, let i be the index of the last node heard by the node u_j.
if u_j has not heard from any node during the last $3(j-i)\frac{n}{\log n}$ rounds. **then**
    Node switches to active state.
else
    Execute the exactly same steps as "Nodes in Active State" and "Nodes in Listening State" (stated in Algorithm 5) executed in Algorithm 4.
end if
Nodes in the Exit State
Nodes that are marked as a *faulty-node* in the faulty-list are moved to the final degree sequence $\mathcal{D}'$.
Send out an *all-okay* message to the nodes in the network in groups of size $O(\log n)$ (see Algorithm 7 for the exact detail) and exit the protocol.
All the non-faulty nodes have the same view of *non-faulty-list*. Therefore, graph is realized locally by the Havel-Hakimi algorithm.

time. A degree-sequence $(d_1, d_2, \ldots, d_n)$ is also given to all the nodes in such a way that each d_i is known to only one node. In this section, we present an algorithm that guarantees that (i) all the non-crashed nodes learn and recreate a list whose size is at least $n - f$, and (ii) this list is the same for all the nodes. This allows the non-crashed nodes to locally realize the overlay graph with the help of Havel-and-Hakimi's algorithm, described in Section 4.3. Our algorithm does not require any knowledge of the number of faulty nodes and IDs of neighboring nodes in the CONGEST model. We use only a few numbers of nodes to propagate the information about the crashed nodes to the other nodes in the network which help to minimize round and message complexity.

4.6.1 Algorithm

The challenging part of designing an efficient algorithm is handling the faulty nodes. A faulty node may crash in some rounds and its message may not reach all the destination nodes in that round. Thus, there might be two sets of nodes with different degree-sequence. This may lead to an incorrect graph realization whose realization is not the same throughout the network. One of the simple ways to solve this problem is as follows: every node u_i sends its ID-degree pair $\langle u_i, d(u_i)\rangle$ and whatever new degree-ID pair received as a message from other nodes to all the other nodes. Since there are n nodes, therefore, each node sends n messages to the n nodes. In this way, we have message complexity $O(n^3)$, and round complexity $O(n)$ when f is unknown to the network. Our main aim is to get the optimal round complexity, i.e., $O(f)$ by keeping the message complexity as small as possible.

The idea is to run the algorithm in three phases, the initialization phase, the performance phase, and the finalization phase. In the initialization phase which consists of only two rounds, in which, each node sends the message as its ID-degree pair twice to other nodes based on the received frequency of messages which can be one or two. The receiver node maintains three lists, faulty-list, final-list, and non-faulty-list. In the performance phase, by sharing the degree of the faulty-list and final-list nodes update the non-faulty-list, and at last non-faulty-list becomes the same across the network (for all non-crashed nodes). In the finalization phase, the network realizes the overlay network based on the received degree-sequence by Havel-and-Havkimi's algorithm.

We will now explain the algorithm in detail, whose terminologies and definitions are summarized in Table 4.2, and the complete pseudocode is given in the Algorithm 9.

Initialization Phase: This phase consists of two rounds. For the first two rounds, each node u_i broadcasts a message which contains its ID along with its degree, i.e., $\langle u_i, d(u_i)\rangle$ to all the nodes. Each node u_i maintains three lists in sorted order (ascending, based on ID), faulty-list (F_{u_i}), final-

list (L_{u_i}) and non-faulty-list ($\mathcal{D}'_{u_i}$) based on the frequency of received messages. Faulty-list (F_{u_i}) maintains the messages (ID and corresponding degree) which were received once, i.e., a list of those nodes which have surely crashed during the initialization phase. On the other hand, final-list (L_{u_i}) maintains the messages (ID and corresponding degree) which were received twice, i.e., a list of those nodes which have not crashed according to u_i. Notice that as u_i knows the corresponding link of the received messages included in the final-list, therefore, u_i can communicate to those particular nodes (if required). Final-list is final in the sense that no new message will be included in the final-list throughout the execution of the algorithm. Non-faulty-list ($\mathcal{D}'_{u_i}$) possesses the messages (ID-degree pair) which were received twice, the degree of the list will be used for the graph realization problem, at last, by using Havel-Hakimi's algorithm. This non-faulty-list will be updated by only including more messages. Each node u_i keeps its message in L_{u_i} and $\mathcal{D}'_{u_i}$ since it has not crashed for the first two rounds. Notice that no message will be removed from the non-faulty-list during the execution of the algorithm. Only faulty-list perform the operation include/remove during the execution of the algorithm. Each u_i performs sorting operations in all the three lists, after an update, in their respective list w.r.t. ID in ascending order.

Algorithm 9 FAULT-TOLERANT GRAPH REALIZATION IN KT_0: CODE FOR A NODE u_i.

Require: A complete n nodes $U = \{u_1, u_2, \ldots, u_n\}$ anonymous network with unique ID. Each node u_i possess a degree of $d(u_i)$ of size $O(\log n)$.

Ensure: A corresponding graph realization that satisfies the condition of distributed graph realization with faults.

INITIALIZATION PHASE:

1: For the first two rounds, each node u_i broadcasts a message which contains its ID along with its degree, i.e., $\langle u_i, d(u_i)\rangle$ to all the nodes.

2: Based on the frequency (once or twice) of received messages, each node u_i maintains the faulty-list F_{u_i}, final-list L_{u_i} and non-faulty-list $\mathcal{D}'_{u_i}$ with received IDs and the corresponding degrees $d(ID)$. If u_i received the message once, then moves the message into F_{u_i} otherwise moves into L_{u_i} and $\mathcal{D}'_{u_i}$. ▷ Message contains $\langle ID, d(ID)\rangle$

3: Each node u_i keeps its message in L_{u_i} and $\mathcal{D}'_{u_i}$.

4: For a message s, s^{ID} is the ID of the message s. $min(L_{u_i})^{ID}$ and $min(L_{u_i})^{d(ID)}$ is the minimum ID and corresponding degree in the list L_{u_i}, respectively. F_{u_i}, L_{u_i} and $\mathcal{D}'_{u_i}$ remain sorted whenever an include/remove operation takes place (in the sorted order w.r.t. their ID).

PERFORMANCE PHASE:

5: Perform performance-phase (u_i).

FINALIZATION PHASE:

6: Every non-crashed node u_i's non-faulty-list $\mathcal{D}'_{u_i}$ have the same view of the degree sequence $\mathcal{D}'$. Therefore, graph realized by all the non-crashed nodes remain same by using the Havel-Hakimi's algorithm.

Algorithm 10 PERFORMANCE-PHASE (u_i)

while $L_{u_i} \neq \phi$ **do**
ACTIVE STATE:
  if $min(L_{u_i})^{ID} = u_i$, or $\exists u_j$ such that $min(L_{u_j})^{ID} = u_i$ **then**
    if $F_{u_i} = \phi$ **then**
      u_i sends $F_{u_i} = \phi$ to all other nodes.
      if u_i receives some message, say s, after sending $F_{u_i} = \phi$ **then**
        u_i includes the unique s (w.r.t. their ID) into F_{u_i}.
      end if
    end if
    while $F_{u_i} \neq \phi$ **do**
      u_i broadcasts the message $\langle min(F_{u_i})^{u_i}, min(F_{u_i})^{d(u_i)} \rangle$ twice to all other nodes. u_i removes the sent message from F_{u_i} and moves the message into $\mathcal{D}'_{u_i}$.
      if u_i receives the message s from u_j such that $s^{ID} < min(F_{u_i})^{ID}$ **then**
        u_i includes those messages in the F_{u_i}.
      end if
    end while
    u_i sends the "agreed" message to all other nodes and moves to *finalization phase*.
  end if
STANDBY STATE:
  if $min(L_{u_i})^{ID} = u_j$ and u_i did not receive any message in last round **then**
    u_i sends $min(L_{u_i})^{ID}$ to u_j and removes the u_j from L_{u_i}.
  end if
  if u_i received the message from u_j such that $F_{u_j} = \phi$ and $F_{u_i} \neq \phi$ **then**
    u_i sends all messages from F_{u_i} to u_j, one-by-one, till receives the messages from u_j. $\triangleright$ u_i does not remove any ID from F_{u_i}
  end if
  if u_i received the message from u_j, say s, such that $s^{ID} > min(F_{u_i})^{ID}$ **then**
    u_i sends all messages whose $s^{ID} > min(F_{u_i})^{ID}$ to u_j, one-by-one, till receives the messages from u_j. $\triangleright$ u_i does not remove any ID from F_{u_i}
  end if
  if u_i receives the message from u_j such that $u_j > min(L_{u_i})^{ID}$ **then**
    u_i removes all such $min(L_{u_i})^{ID}$, iteratively.
  end if
  if u_i receives the message from u_j once, say s, and $s \notin F_{u_i}$ **then**
    u_i includes s into F_{u_i}.
  end if
  if u_i receives the message from u_j twice, say s, and $s \in F_{u_i}$ **then**
    u_i removes s from F_{u_i}.
    if $s \notin \mathcal{D}'_{u_i}$ **then**
      u_i includes s in $\mathcal{D}'_{u_i}$.
    end if
  end if
  if u_i receives the message "agreed" from any node u_j **then**
    u_i sends the message "agreed" to all other nodes and moves to *finalization phase*.
  end if
end while

Terminology at a Glance for a node u_i	
Terminology	Definition
Faulty-list (F_{u_i})	List of known faulty IDs (and their corresponding degrees) at u_i.
Final-list (L_{u_i})	List of IDs (and their corresponding degrees) that sent their messages in the first two rounds at u_i and have not crashed as per u_i.
Non-faulty-list ($\mathcal{D}_{u_i}$)	List of IDs (and their corresponding degrees) which were received/sent twice from/to u_i. In *finalization phase*, contains the final degree-sequence used for graph realization at u_i.
Standby State	A node is in the Standby state if it is waiting for its turn (to broadcast entries from its faulty-list) or to receive an "agreed" message.
Active State	A node is in the active state if it is broadcasting from its faulty-list.
"agreed"	Node u_i received "agreed" message then u_i becomes agree for the *finalization phase*. u_i broadcasts the "agreed" message and moves to the *finalization phase*.

Table 4.2: Terminology and their definition used throughout the algorithm 9.

Performance Phase: In this phase, nodes send/receive the messages based on the received messages and the status of the F_u, L_u, and $\mathcal{D}'_u$. We study the algorithm from the perspective of some node u_i. We consider two states of the node u_i: (i) active state and (ii) standby state. In the case of an active state, node u_i which possesses the minimum ID among the non-crashed nodes broadcast the message to all other nodes. While in the case of standby state, non-crashed node u_i sends the messages to a particular node based on the condition that arises. This is done by performing the following steps iteratively.

Active State: Node u_i reaches in active state if u_i has received the message from some node u_j whose minimum ID's entry in the L_{u_j} is u_i or L_{u_i} has the minimum ID as u_i then u_i checks its F_{u_i}. There can be two conditions with F_{u_i} either F_{u_i} is empty or not. In case, if F_{u_i} is empty then u_i asks all other nodes to send messages from their respective faulty-list (if any). If F_{u_i} is empty (did not receive the message in the next round) then u_i sends the message "agreed" to all other nodes and moves to the finalization phase. On the other hand, if u_i received some messages from this call then u_i includes those messages in the F_{u_i}. On the assumption, If node u_i has not moved

to the finalization phase then there might have arisen a second condition, i.e., F_{u_i} is non-empty. In that event, u_i sends the message from F_{u_i} twice and asks whether they have any message in their faulty-list whose ID is less than what they just received. If other nodes have such messages in their respective faulty-list then they send all these messages one by one to u_i. u_i includes all these coming messages into F_{u_i} and sends twice to all other nodes. In between, all other nodes send their appropriate messages from faulty-list (messages whose IDs are less than what they just received, only once to u_i) only if they are receiving the messages from u_i. In case, u_i is not sending the message in some rounds which signifies that u_i has crashed, therefore, there is no need to send the message to u_i. If F_{u_i} becomes empty at last and u_i has not crashed then u_i sends the "agreed" message to all other nodes and moves to the finalization phase.

Standby State: In this state, node u_i communicates to the particular node, say u_j. For a node u_i when the minimum ID of L_{u_i} is u_j and u_i did not receive any message in the previous round then u_i asks u_j for initiation and removes the u_j from L_{u_i}. There is a possibility that u_j has not crashed since it sent the message twice in the initialization phase. In case, $F_{u_i} \neq \phi$ and u_i receives the messages such that $F_{u_j} = \phi$ then u_i sends all messages from (F_{u_i}) to u_j, one-by-one, till receives the message from u_j. Also, if u_i received the message from u_j whose ID is greater than the minimum of F_{u_i}'s ID or then u_i sends all those messages whose ID is greater than (F_{u_i}) to u_j, one-by-one, till receives the message from u_j. So that u_j can convey those messages (which were not received by all the nodes) to all other nodes. Notice that u_i does not remove any of these IDs from F_{u_i} since there might be the case that u_j crashes without conveying the message. Therefore, u_i removes only those messages from F_{u_i} which are received twice by the u_i. On the other hand, if u_i receives the message from u_j where u_j is greater than the minimum of L_{u_i} then u_i removes all such messages from L_{u_i}, iteratively. Since those nodes have already crashed (otherwise u_j has not been active to send the message) and there is no need to communicate with them in coming rounds (in case, u_j crashed). Supposing u_i receives the message from u_j once, say s, and $s \notin F_{u_i}$ then u_i includes s into F_{u_i}. In case, if u_i receives the message from u_j twice and $s \in F_{u_i}$ then u_i removes s from F_{u_i}. If $s \notin \mathcal{D}'_{u_i}$ and received twice then u_i includes s in $\mathcal{D}'_{u_i}$. Node u_i moves to the finalization phase of the algorithm if u_i has received the "agreed" message from any node u_j. In that situation, u_i sends the message "agreed" to all the nodes and moves to the finalization phase of the algorithm. The "agreed" message conveys the information to other nodes that there does not exist any node which possesses a non-empty faulty-list or a different view of non-faulty-list.

Finalization Phase: In this phase, each non-crashed node u_i's non-faulty-list $\mathcal{D}'_{u_i}$ has the same view of degree sequence ($\mathcal{D}'$) throughout the network. Therefore, the graph realized by all the non-crashed nodes remains the same by using the Havel-Hakimi's algorithm 1.1.

The above three phases are performed till the nodes receive the message "agreed" and the algorithm terminates. In the end, all the non-faulty nodes have the same degree sequence in their non-faulty-list, which they realize by Havel-Hakimi's algorithm.

We will now show the correctness of the algorithm with the help of Lemma 4.7 that the final degree sequence $\mathcal{D}'$ of all the non-crashed nodes is the same. Thus, the algorithm correctly solves the distributed graph realization with faults in the Congested Clique in the KT_0 model. Further, we analyze the time and the message complexity by using Lemma 4.8, Lemma 4.9 and Lemma 4.10.

Lemma 4.7. *If there exist some non-faulty nodes u_i and u_j such that $\mathcal{D}'_{u_i} - \mathcal{D}'_{u_j} \neq \phi$ at some point, then in the finalization phase of the algorithm there exist $\mathcal{D}'_{u_i} - \mathcal{D}'_{u_j} = \phi$.*

Proof. Let us suppose at some point there exists some message s such that $\mathcal{D}'_{u_i} - \mathcal{D}'_{u_j} = s$. This implies that the sender of s crashed in the second round such that u_i received the message twice but not u_j. Now, if non-faulty node u_j (or some node which has s in faulty-list) becomes the minimum ID in its L_{u_j} or some other node's final-list (becomes active node [2]) then u_j will eventually send the s to all other nodes twice. Therefore, s will be part of $\mathcal{D}'$ across the network. Similarly, if some node u_i (or some node which does not possess s in faulty-list) becomes active node then u_i might send higher ID message to u_j or send $F_{u_i} = \phi$ to u_j and asks about faulty value. In that case, u_j sends the s and u_i broadcasts the s to all other nodes. Therefore, s becomes the part of $\mathcal{D}'$ across the network. Hence, the lemma. □

Lemma 4.8. *In a non-faulty setup [3], round complexity is $O(1)$ and message complexity is $O(n^2)$.*

Proof. In the *initialization phase*, all the nodes send their respective ID and corresponding degree twice successfully. Therefore, the faulty-list across the network remains empty. In the *performance phase*, the node with minimum ID, say u_i, asks all the nodes about the status of their faulty-list and waits for a round. In parallel, other nodes are also asking u_i to be active. Further, after waiting for a round u_i broadcasts the "agreed" message and reaches in the *finalization phase*. In the very next round, other nodes also reach *finalization phase*. In this scenario, the algorithm is executed in constant rounds and each round takes $O(n^2)$ a message. Hence, the lemma. □

Lemma 4.9. *All the faulty-nodes f cost $O(nf)$ messages and $O(f)$ rounds extra as compared to non-faulty setup.*

Proof. A faulty node u_i may not follow the protocol during the crash. It may deviate in, mainly, two phases: (i) *initialization phase* or (ii) *performance phase*. In the initialization phase, node

[2] Node which is in active state (standby state) considered as active node (standby node).
[3] A non-faulty setup is the model in which all the nodes are non-faulty.

u_i may crash during the broadcasts of the message. If it crashed in the first round or second round then some nodes possess node u_i in their faulty-list. During the *performance phase*, some non-faulty nodes broadcast this value twice from its faulty-list. Therefore, this faulty-node may cause 2 extra rounds as compared to non-faulty setup (Lemma 4.8). Further, if a faulty node crash in *performance phase* during the broadcasts of the message then there might be some nodes that do not receive the message twice which would be broadcasted by some non-faulty node again. Therefore, this faulty-node may also cause 2 extra rounds as compared to non-faulty setup (Lemma 4.8). From the above discussion, we can see that one faulty node can cause an extra cost of $O(1)$ rounds and $O(n)$ messages (due to broadcast). Therefore, the overall extra messages and rounds cost of the algorithm for f faulty nodes are $O(nf)$ and $O(f)$, respectively. □

Lemma 4.10. *The time and message complexity of the Algorithm 9 is $O(f)$ and $O(n^2)$, respectively.*

Proof. From the Lemma 4.8, in non-faulty setup, we have the round complexity $O(1)$ and message complexity $O(n^2)$. On the other hand, from the Lemma 4.9, we have the extra cost compared to non-faulty setup is $O(f)$ rounds and $O(nf)$ message. Therefore, we can conclude the round complexity is $O(1) + O(f) = O(f)$ and message complexity is $O(n^2) + O(nf) = O(n^2)$, as $f < n$. □

Thus, we get the following main result of fault-tolerant graph realization.

Theorem 4.6. *Consider an n-node Congested Clique with KT_0 model, in which $f < n$ nodes may crash arbitrarily at any time. Given an arbitrary n-length degree sequence $\mathcal{D} = (d_1, d_2, \ldots, d_n)$ as an input such that each d_i is only known to one node in the clique. Then there exists a deterministic algorithm that solves the fault-tolerant graph realization problem in $O(f)$ rounds and using $O(n^2)$ messages such that f is unknown to the network.*

4.6.2 Lower Bound

Recall that our network is anonymous, i.e., a node does not know the IDs of its neighbors initially. This model is known as KT_0; on the other hand, in the KT_1 model, nodes know their neighbors. Thus, KT_0 model is a weaker model than KT_1 model, i.e, KT_1 model has some extra information regarding the neighbors' IDs as compared to KT_0. Therefore, the algorithm which can solve the graph realization problem in a Congested Clique in KT_0 model will also solve the problem in KT_1 model with matching complexity. By using the same line of argument, in the case of lower bound

KT_1's lower bound (shown in [94]) is the trivial lower bound for KT_0 model. Therefore, we also have the following results in KT_0 model.

Theorem 4.7. *Any algorithm that solves the distributed graph realization with f crash failures in an n-node Congested Clique requires $\Omega(f)$ rounds in some admissible execution.*

Theorem 4.8. *In the* CONGEST *model, any algorithm that solves the distributed graph realization problem of n node network (with or without faults) requires $\Omega(n^2)$ messages in some admissible execution.*

Notice that Algorithm 9 is, simultaneously, tight with respect to the time complexity and message complexity.

4.7 Conclusion

In this chapter, we studied the round and message complexity of the graph realization problem in the Congested Clique with faults in the KT_1 as well as KT_0 model and provided an efficient algorithm for realizing overlays for a given degree sequence. Our algorithms are simultaneously optimal in both the round and the message complexity. Further, we also show how the algorithm may be adapted in the KT_1 model, to a setting in which nodes are allowed to send (and receive) a limited number of messages per round.

Given the relevance of graph realization techniques in overlay construction and the presence of faulty nodes in peer-to-peer networks, we believe there can be several gripping questions to explore in the future. Such as:

(1) Does the message and round complexity $O(n^2)$ and $O(f)$, respectively, hold for the omission failure[4]? Also, what would be the nontrivial lower bound for the omission failure?

(2) It would be entrancing to define and analyze the graph realization problem in the presence of Byzantine faults. Since a Byzantine node can behave arbitrarily like sending the wrong message, sending a message to some nodes, or not sending the message in some rounds. Therefore, the graph realization problem needs to be defined carefully.

(3) Our work and existing work on distributed graph realization consider a clique network [12]. It would be interesting to study the problem in general networks.

[4]A faulty node crashes or omits to send messages that it supposed to send (send omission) or omits to receive messages that it supposed to receive (receive omission).

Chapter 5

Optimal Algorithm for Deterministic Leader Election in Diameter-Two Networks

In this chapter, we focus on the leader election problem in diameter-two networks [1]. Chatterjee et al. [31] recently studied the leader election in diameter-two networks. They presented a $O(\log n)$-round deterministic implicit leader election algorithm which incurs optimal $O(n \log n)$ messages, but a drawback of their algorithm is that it requires knowledge of n. An important question – whether it is possible to remove the assumption on the knowledge of n was left open in their paper. Another interesting open question raised in their paper is whether *explicit* leader election can be solved in $\tilde{O}(n)$ messages deterministically. In this chapter, we give an affirmative answer to them. Further, we solve the *broadcast problem*, another fundamental problem in distributed computing, deterministically in diameter-two networks with $\tilde{O}(n)$ messages and $\tilde{O}(1)$ rounds without the knowledge of n. In fact, we address all the open questions raised by Chatterjee et al. for the deterministic leader election problem in diameter-two networks.

First, we present a deterministic *explicit* leader election algorithm which takes $O(\log \Delta)$ rounds and $O(n \log \Delta)$ messages, where n in the number of nodes and Δ is the maximum degree of the network. The algorithm works without the knowledge of n. The message bound is tight due to the matching lower bound, showed Chatterjee et al. [31].

Then we show that *broadcast* can be solved deterministically in $O(\log \Delta)$ rounds using $O(n \log \Delta)$ messages. More precisely, a broadcast tree can be computed with the same complexities and the depth of the tree is $O(\log \Delta)$. This also does not require the knowledge of n.

5.1 Introduction

In the four decades since its inception, leader election has remained a well-explored and fundamental problem in distributed networks [100, 101, 116]. The basic premise of leader election is simple: given a group of n nodes, a unique node is elected as a leader (where n denotes the number

[1] These findings are based on joint work with Anisur Rahaman Molla and Sumathi Sivasubramaniam (which appeared in International Conference on Algorithms and Complexity 2023) and contains material from [95].

of nodes in the network). Depending on the nodes' knowledge of the leader, there are two popular versions. In the first version (known as the *implicit* leader election), the non-leader nodes are not required to know the leader's identity; it is enough for them to know that they are not the leader. The *implicit* leader election is quite well studied in literature [11, 92, 96, 97, 106]. In the other version (known as *explicit* leader election), the non-leader nodes are required to learn the leader's identity. The implicit version of the leader election is the generalized version of the (explicit) leader election. Clearly, there is a lower bound of $\Omega(n)$ for message complexity in the explicit version of the problem. In this chapter, we study the explicit version of the problem. In particular, we show an improvement on the existing deterministic solution for the implicit leader election algorithm presented in [31] and provide an algorithm for turning the implicit leader election explicit without any additional overhead on messages.

Leader election has been studied extensively with respect to both message and round complexity in various graph structures like rings [101, 136], complete graphs [4, 14, 69, 83, 85, 86, 130], diameter-two networks [31] etc., as well as in general graphs [54, 60, 97, 106, 117] [2]. Earlier works were primarily focused on providing deterministic solutions. However, eventually, randomized algorithms were explored to reduce mainly the message complexity (see [14, 60, 96, 97] and the references therein). Kutten et al. gave the fundamental lower bound for leader election in general graphs with $\Omega(m)$ message complexity and $\Omega(D)$ round complexity [96], where m is the number of edges and D is the diameter of the graph. This bound is applicable for all graphs with diameters greater than two, whether the algorithm is deterministic or randomized. For the clique, recently a tight message lower bound of $\Omega(n \log n)$ is established by Kutten et al. [98] for the deterministic algorithms under simultaneous wake-up of the nodes. The same lower bound was shown earlier by Afek and Gafni (1991) [4], but assumes adversarial wake-up. Table 5.1 presents an overview of the results (deterministic). Recently, diameter-two networks were explored, and the message complexity was settled by providing a deterministic algorithm with $O(n \log n)$ message complexity [31].

Our work is closely related to the work by Chatterjee et al. [31]. In their work, the authors studied leader election (the *implicit* version) in diameter-two networks. They presented a deterministic algorithm with $O(n \log n)$ message complexity and $O(\log n)$ round complexity. Crucially, their algorithm requires prior knowledge of the size of the network, n. In comparison to this, our algorithm elects a leader *explicitly* without prior knowledge of n. Our algorithm uses $O(n \log \Delta)$ messages and finishes in $O(\log \Delta)$ rounds, where Δ is the maximum degree of the graph (see, Table 5.2). In addition to this, we show how to leverage the edges used during the leader election

[2] We interchangeably use the word "graph" and "network" throughout the chapter.

protocol to create a broadcast tree for the diameter-two graphs with a message and round complexity of $O(n \log \Delta)$ and $O(\log \Delta)$, respectively. Computing a broadcast tree efficiently is another fundamental problem in distributed computing. A broadcast tree can be used as a subroutine to many distributed algorithms which look for message efficiency. Finding a deterministic $\tilde{O}(n)$-message and $\tilde{O}(1)$-round broadcast algorithm in diameter-two networks was also left open in [31]. We have addressed it.

DETERMINISTIC (EXPLICIT) LEADER ELECTION RESULTS			
Paper	Message Complexity	Round Complexity	Graph of Diameter
Afek-Gafni [4]	$O(n \log n)$	$O(\log n)$ *	$D = 1$
Kutten et al. [98]	$\Omega(n \log n)$	$\Omega(1)$	$D = 1$
This chapter	$O(n \log \Delta)$	$O(\log \Delta)$	$D = 2$
Chatterjee et al. [31]	$\Omega(n \log n)$	$\Omega(1)$ **	$D = 2$
Kutten et al. [96]	$O(m \log n)$	$O(D \log n)$	$D \geq 3$
Kutten et al. [96]	$\Omega(m)$	$\Omega(D)$	$D \geq 3$

Table 5.1: Best known deterministic leader election results on networks with different diameters. Since $\Delta = \Omega(\sqrt{n})$ in diameter-two graphs, $\log \Delta = O(\log n)$, see the Remark 3 below. So our upper bound does not violate the message lower bound in [31]. * 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 [4]. ** $\Omega(1)$ is a trivial lower bound.

Chapter Organization: In the rest of this section 5.1, we state our results. In Section 5.2, we present our model and definitions. We briefly introduce various related works in Section 5.3. We present our algorithms for deterministic leader election and broadcast tree formation in Section 5.4. We conclude the chapter in Section 5.5.

5.1.1 Our Results

Our work focuses on the deterministic leader election in diameter-two networks without the knowledge of the number of nodes. Apart from this, by leveraging the leader election protocol, we show

that *broadcast* can be solved deterministically, matching the complexity of the leader election algorithm. Specifically, we have the following results.

1. We present a deterministic *explicit* leader election algorithm which takes $O(\log \Delta)$ rounds and $O(n \log \Delta)$ messages, where n in the number of nodes and Δ is the maximum degree of the network. The algorithm works without the knowledge of n. The message bound is tight due to the matching lower bound, showed by Chatterjee et al. in [31].

2. We show that *broadcast* can be solved deterministically in $O(\log \Delta)$ rounds using $O(n \log \Delta)$ messages. More precisely, we show that a broadcast tree, of depth at most $O(\log \Delta)$ can be computed with the same complexities.

5.2 Model and Definition

Our model is similar to the one in [31]. We consider the distributed network to be an undirected graph $G = (V, E)$ of n nodes and diameter $D = 2$. Each node has a unique ID of size $O(\log n)$ bits. The model is a *clean network model* in the sense that the nodes are unaware of their neighbors' IDs initially, also known as KT_0 model [118]. The network is synchronous. The nodes communicate via passing messages in a synchronous round. We limit each message to be of size at most $O(\log n)$ bits as in the CONGEST communication model in distributed networks [118]. In each round, nodes may send messages, receive messages and perform some local computation. The round complexity of an algorithm is the total number of rounds of communication taken by the algorithm before termination. The message complexity is the total number of messages exchanged in the network throughout the execution of the algorithm. Throughout this chapter, we assume that all nodes are awake initially and simultaneously start executing the algorithm.

We will now formally define the implicit and explicit versions of leader election in our model.

Definition 5.1 (**Implicit Leader Election**). *Consider an n-node distributed network. Let each node maintain a state variable that can be set to a value in $\{\perp, NONELECTED, ELECTED\}$, where $\perp$ denotes the 'undecided' state. Initially, all nodes set their state to $\perp$. In the implicit version of leader election, it requires that exactly one node has its state variable set to $ELECTED$ and all other nodes are in state $NONELECTED$. The unique node whose state is $ELECTED$ is the leader.*

Definition 5.2 (**Explicit Leader Election**). *Consider an n-node distributed network. Let each node maintain a state variable that can be set to a value in $\{\perp, NONELECTED, ELECTED\}$,*

DETERMINISTIC LEADER ELECTION IN DIAMETER-TWO GRAPHS				
Paper	Message Complexity	Round Complexity	Type	Knowledge of n
Chatterjee et al. [31]	$O(n \log n)$	$O(\log n)$	Implicit	YES
This chapter	$O(n \log \Delta)$	$O(\log \Delta)$	Explicit	NO

Table 5.2: Comparison of the current chapter to the state-of-the-art.

where $\perp$ denotes the 'undecided' state. Initially, all nodes set their state to $\perp$. In the explicit version of leader election, it requires that exactly one node has its state variable set to $ELECTED$ and all other nodes are in state $NONELECTED$. Further, the $NONELECTED$ nodes must know the identity of the node, whose state is $ELECTED$, the leader.

5.3 Related Work

In 1977, the leader election problem was introduced by Le Lann in the ring network [101]. Since then the problem has been studied extensively in different settings. The leader election problem has been explored in both implicit and explicit versions over the years [74, 106, 117, 97, 60, 90] for a variety of models and settings, and for various graph topologies such as cliques, cycles, mesh, etc., (see [74, 92, 72, 79, 117, 126, 132, 123] and the references therein for more details). In general, the implicit leader election suffices for most networks.

Both deterministic and randomized solutions exist for leader election. For the randomized case, for complete graphs, Kutten et al. [97] showed that $\tilde{\Theta}(\sqrt{n})$ is a tight message complexity bound for randomized (implicit) leader election. For any graph with a diameter greater than 2, the authors in [97] showed that $\Omega(D)$ is a lower bound for the number of rounds for leader election using a randomized algorithm (they also showed a lower bound for the message complexity, $\Omega(m)$). Recently, Chatterjee et al., [31] showed a lower bound of $\Omega(n)$ for the message complexity of randomized leader election in diameter-two graphs.

In the deterministic case, it is known that $\Theta(n \log n)$ is tightly bound on the message complexity for complete graphs [4, 98]. This tight bound also carries over to the general case as seen from [4, 84, 86]. In our work, we restrict our model to graphs of diameter-two. For diameter-two graphs, Chatterjee and colleagues provide a $O(\log n)$ round algorithm that uses $O(n \log n)$ mes-

sages. However, their algorithm requires knowledge of n, our algorithm provides an algorithm that requires no prior knowledge of n and runs in $O(\log n)$ rounds with $O(n \log n)$ message complexity.

5.4 Deterministic Leader Election in Diameter-Two Networks

We present a deterministic (explicit) leader election algorithm for diameter-two networks with n nodes in which the value of n is unknown to nodes in the network. In this section, we answer several questions raised in [31]. Specifically, we address the following: (i) Can explicit leader election be performed in $\tilde{O}(n)$ messages in diameter-two graphs deterministically? (ii) Given the leader election algorithm, can broadcast can be solved deterministically in diameter-two graphs with $\tilde{O}(n)$ message complexity and $O(\text{polylog}\, n)$ rounds if n is known, and crucially (iii) "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." In this section, we solve the explicit leader election with $\tilde{O}(n)$ message complexity, along with that our algorithm solves the explicit leader election without the knowledge of n; thus addressing the questions (i) and (iii). We further present a solution for the question (ii) that too without the knowledge of n.

5.4.1 Algorithm

Our algorithm is inspired from the work done by Chatterjee et al. [31]. They presented an algorithm for implicit leader election that ran in $O(\log n)$ rounds with $O(n \log n)$ message complexity (with the knowledge of n). Our Algorithm 11, achieves somewhat better result without the knowledge of n and also elects the leader explicitly. We assign the highest priority to the degree unlike the ID (as in [31]) and based on that calculate the bound on the highest degree of the neighbouring nodes. This technique eventually helps in termination of the algorithm without knowing n. Rest of the results are the follow-up after electing the leader. We also use the Lemmas and Algorithm's steps from [31] for sake of compilation.

As mentioned earlier (in Section 5.2), each node has a unique ID. For any node $v \in V$, let us denote the degree of v by d_v and the ID of v by ID_v. The *priority* $\mathcal{P}_v$, of node v, is a combination of the degree and ID of the node v such that $\mathcal{P}_v = \langle d_v, ID_v \rangle$. The leader is elected based on the priority, which is decided by the degree of the node. In the case of a tie, the higher ID gets the higher priority. Essentially, the node with the highest priority becomes the leader.

Our algorithm runs in two phases of $O(\log d_v)$ rounds each. In the first $O(\log d_v)$ rounds, we

eliminate as many invalid candidates as possible. In the second phase, all candidates except the actual leader are also eliminated, culminating in the election of a unique leader.

Detailed description of the algorithm:

Initially, every node is a "candidate" and has an "active" status. Each node v numbers its neighbors from 1 to d_v arbitrarily, denoted by $w_{v,1}, w_{v,2}, \ldots, w_{v,d_v}$. For the first $i = 1$ to $\log d_v$ rounds, if v is active, then node v sends a message containing $\mathcal{P}_v$ to its neighbors $w_{v,2^{i-1}}, \cdots, w_{v,\min\{d_v,2^i-1\}}$. If v encounters a priority higher than its own from its neighbors (either because a neighbor has a higher priority or has heard of a node with higher priority) then v becomes "inactive" and "non-candidate". That is, v does not send any further messages to its neighbors containing v's priority. Although, v may send a higher priority message based on the received message's priority (explained later). Let L_v denotes the ID of the current highest priority node known to v. At the beginning of the execution, L_v is simply $\mathcal{P}_v$. If at the end of the first $\log d_v$ rounds, $L_v = \mathcal{P}_v$ then v declares itself leader temporarily. Further, v waits for $\log d_v$ rounds. If at the end of $\log d_v$ rounds v is still the candidate node (v has not heard from a node about the higher priority) then v becomes the leader.

There are two major phases to the algorithm. For the first $\log d_v$ rounds, we eliminate as many invalid candidates (the node which has encountered higher priority node) as possible, as follows. N_v contains the ID of the neighbor that informed v about the current highest priority. As mentioned before, L_v contains the current highest priority known to v. Let χ_v denote the (possibly empty) set of v's neighbors from whom v has received messages in a round during this phase, and $\mathcal{P}(\chi_v)$ be the set of $\mathcal{P}$s sent to v by the members of χ_v such that $\mathcal{P}_u$ be the highest $\mathcal{P}$ in $\mathcal{P}(\chi_v)$. If $\mathcal{P}_u$ is higher than that of L_v then v stores the highest priority seen so far in L_v. Further, v informs N_v about $L_v = \mathcal{P}_u$, i.e., the highest $\mathcal{P}$ it has seen so far. This particular step exploits the neighborhood intersection property to ensure that information about higher priority nodes is disseminated quickly. Then v updates N_v. Finally, v tells every member of χ_v about L_v, i.e., the highest $\mathcal{P}$ it has seen so far. If $L_v \neq P_v$ then v becomes "inactive" and "non-candidate". Notice that an "inactive" and "non-candidate" node v only disseminates the information of higher priorities it hears, to N_v.

At the end of the first $\log d_v$ rounds, we begin the final phase of the election. If v is still the candidate node then v waits for $\log(d_v)$ rounds. Furthermore, if v does not receive any higher priority message then v declares itself as the leader and informs its neighbors. Then each neighbor of v, say u, informs their neighbors about the election of v via a set of Ψ_u nodes. On the other hand, if there exists a node whose priority is higher than the priority of v then v gets to know about the leader and informs all the nodes to whom v has communicated (so far) about the leader's ID

(that is the set Ψ_v) and exits. Hence, All the nodes elect the same leader whose priority is the highest. Our claim is that given certain properties of the degree (see Lemma 5.1) we can guarantee that the second phase of waiting for $\log \Delta$ rounds eliminates all but a unique candidate, which then becomes the leader.

Now, we would discuss some important lemmas and the correctness of the algorithm. Finally, we conclude the result in Theorem 5.1.

Lemma 5.1. *Let v be a node whose degree, d_v, is the highest among its neighbors and Δ is the maximum degree of the graph. There does not exist any diameter-two graph with n nodes ($n > 4$) such that $\Delta > d_v^2$.*

Proof. For a node v, all nodes are at most 2 hop distance away from v, since the diameter of the graph is 2. Node v has degree d_v and its neighbors have degree at most d_v, by assumption. This gives an upper bound on n, that is, $n \leq d_v(d_v - 1) + 1$, because each of the d_v neighbors can have at most other $d_v - 1$ neighbors (excluding v) each, and by the distance assumption, there are no other nodes in the graph. Also, Δ can be at most $n-1$. Therefore, $\Delta < n < d_v^2 + 1$. Consequently, $d_v^2 > \Delta$. Hence, the lemma. □

Remark: 3. *It is clear that there does not exist any diameter-two network whose nodes are neither connected to v nor its neighbor. Therefore, $d_v^2 \geq n$. This implies $d_v \geq \sqrt{n}$. Hence, $\Delta \geq \sqrt{n}$.*

Lemma 5.2. *Algorithm 11 solves the leader election in $O(\log \Delta)$ rounds, where Δ is the maximum degree of the graph.*

Proof. A candidate node v becomes the leader if its priority is the highest among its neighbors (Line 22). From Lemma 5.1, we know that $\Delta < d_v^2$. Therefore, the node v with degree d_v waits for $\log d_v$ rounds, in that time, the node with degree Δ informs about its priority to v (if any) and v becomes inactive. Otherwise, v considers d_v as Δ and informs all its neighbors about its election. v's neighbor further conveys the message to all other nodes in $\log \Delta$ rounds. Therefore, the round complexity of the algorithm is $O(\log \Delta)$. □

For the message complexity analysis, we adapt a couple of results from [31], since our algorithm (Algorithm 11) uses a similar approach to keep a node active. In particular, we use the Lemma 11 and Lemma 12 from [31], which used ID of the nodes to take a decision on the "active" or "inactive" nodes whereas our algorithm uses priority (which depends on degree and ID). Hence, the results also apply to our algorithm. The following two lemmas are adapted from Lemma 11 and Lemma 12 in [31].

Algorithm 11 DETERMINISTIC-LEADER-ELECTION: CODE FOR A NODE v

Require: A two-diameter connected anonymous network. Each node possesses a unique ID.
Ensure: Leader Election.

1: v becomes a "candidate" and "active".
2: Let $\mathcal{P}_v = \langle d_v, ID_v \rangle$ be the priority of v. Priority is determined by degree, the node with the higher degree (d_v) has higher priority. The node's ID is used to break any ties.
3: $L_v \leftarrow \mathcal{P}_v$ ▷ L_v is the current highest priority known to v.
4: $N_v \leftarrow \mathcal{P}_v$ ▷ N_v is the neighbor which informed about L_v.
5: v creates an arbitrary assignment of its neighbors based on its degree (from 1 to d_v) which are called $w_{v,1}, w_{v,2}, \cdots, w_{v,d_v}$ respectively.
6: **for** rounds $i = 1$ to $\log d_v$ **do**
7: **if** v is active **then**
8: v sends a "probe" message containing its priority $\mathcal{P}$ to its neighbors $w_{v,2^{i-1}}, \cdots, w_{v,\min\{d_v,2^i-1\}}$.
9: **end if**
10: Let χ_v be the possibly empty subset of $v's$ neighbors from which v received messages in this round.
11: Let $\Psi_v = \bigcup_1^i \chi_v$.
12: Let $\mathcal{P}(\chi_v)$ be the set of $\mathcal{P}$s sent to v by the members of χ_v.
13: Let $\mathcal{P}_u$ be the highest $\mathcal{P}$ in $\mathcal{P}(\chi_v)$.
14: **if** $\mathcal{P}_u > L_v$ **then**
15: $L_v \leftarrow \mathcal{P}_u$
16: v tells N_v about $L_v = \mathcal{P}_u$, i.e., the highest $\mathcal{P}$ it has seen so far.
17: $N_v \leftarrow x$. ▷ v remembers neighbor who told v about L_v.
18: v becomes "inactive" and "non-candidate".
19: **end if**
20: v tells every member of χ_v about L_v, i.e., the highest $\mathcal{P}$ it has seen so far.
21: **end for**
22: **if** $L_v = \mathcal{P}_v$ **then**
23: v waits for $\log(d_v)$ rounds. If at the end of $\log(d_v)$ rounds, $L_v = \mathcal{P}_v$ then v declares itself as a leader and informs all the neighbors as well as exits the protocol.
24: **end if**
25: **if** v knows about the leader and v is not the leader **then**
26: Let Φ_v be the set of neighbors of v to whom v sent the messages before knowing about the leader.
27: Let $\Psi_v = \Psi_v \bigcup \Phi_v$.
28: v informs Ψ_v about the leader's ID and exit.
29: **end if**
30: All the nodes elect the same leader whose priority is the highest.

Lemma 5.3 ([31]). *"At the end of the 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 14 of Algorithm 11 has not so far been satisfied for v, which in turn implies that $\mathcal{P}_v > \mathcal{P}_{w_{v,j}}$ 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.,

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

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

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

(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. Case-by-case analysis can be found in [31]." □

Lemma 5.4 ([31]). *In round i, Algorithm 11 transmits at most $3n$ messages in the for loop (from Line 6 to Line 21).*

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 Lines16 and 20 of Algorithm 11). Thus, in round i, each active node contributes to, directly or indirectly, at most $3 \cdot (2^i - 1)$ messages. The result immediately follows from Lemma 5.3. □

Lemma 5.5. *The message complexity of the Algorithm 11 is $O(n \log \Delta)$.*

Proof. Each round transmits at most $3n$ messages (Lemma 5.4) and the execution of the Algorithm 11 (from Line 6 to Line 21) takes place in $O(\log \Delta)$ rounds (Lemma 5.2). Further, the leader informs about its election via Ψ edges which are $O(n \log \Delta)$. Therefore, the total number of messages transmitted throughout the execution are: $3n \cdot O(\log \Delta) + O(n \log \Delta) = O(n \log \Delta)$. □

Correctness of the Algorithm: In this, we show that all the nodes agree on a leader and the leader is unique. First, we show that all the nodes agree on a leader. If a node v is still a candidate node at the end of the first phase, then it must have both i) explored all its neighbors and ii) never encountered a priority higher than its own. Thus, it can declare itself a leader after waiting for $\log d_v$ rounds. Note that a waiting period of $\log d_v$ is enough because from Lemma 5.1 we know that $\Delta < d_v^2$. This guarantees that the highest degree is made leader.

Now, we show that the known leader is unique. If not, then suppose there exist two nodes u and v such that u agrees on a leader l_1 and v agrees on a leader l_2. From algorithm 11, l_1 should have the highest priority in its neighbors and similarly, l_2 should have the highest priority in its neighbors. Since it is a diameter two graph, therefore, there should be at least one node common among l_1 and l_2. Therefore, both nodes can not have the highest priority among their neighbors, which is a contradiction. Therefore, we can say all the nodes agree on the unique leader.

From the above discussion, we conclude the following result.

Theorem 5.1. *There exists a deterministic (explicit) leader election algorithm for n-node anonymous networks with diameter two that sends $O(n \log \Delta)$ messages and terminates in $O(\log \Delta)$ rounds, where Δ is the maximum degree of the network.*

Remark: 4. *The implicit deterministic leader election algorithm presented in [31] can be converted to an explicit leader election algorithm in the same way as done in Algorithm 11.*

5.4.2 Broadcast Tree Formation

In Algorithm 11, the nodes agree on the leader explicitly. In this section, we exploit the edges used during the leader election algorithm (Algorithm 11) and create a broadcast tree of height $O(\log \Delta)$ (Algorithm 12). This also allows to reduce the message complexity. The process is simple. The leader, say ℓ, initiates the flooding process by broadcasting its ID to its neighbors, forming the root of the tree T. All of its neighbors become a part of T. At any point in the algorithm, the leaves of T do the following. Let v be a leaf in T in some round. In that round, v sends its own ID to the nodes in Ψ_v (used in Algorithm 11). Non-tree nodes which receive an ID v earlier become a part of T with v as its parent. If a non-tree node receives multiple messages, then it chooses the higher ID as its parent. The algorithm ends when all nodes have become a part of T. Note that since only the leaves send out messages in each round and each node (except the root node, i.e., leader node) possesses only one parent, we avoid the creation of cycles.

Let us now show some important lemmas which support the correctness of the algorithm. In particular, Lemma 5.7 shows Algorithm 12 forms a tree of height $O(\log \Delta)$. The round complexity

and message complexity of the Algorithm 12 is shown by Lemma 5.6 and Lemma 5.8, respectively. Finally, we conclude with the message and round complexity as well as the height of the tree in Theorem 5.2.

Algorithm 12 BROADCAST-TREE-FORMATION

Require: A diameter-2 connected network graph G in which each node possess unique ID.

Ensure: Tree Structure T.

1: First run Algorithm 11 to elect the leader ℓ. Each node also keeps track of its Ψ_v (created during the course of the algorithm).
2: ℓ becomes root of T. ℓ then broadcasts its ID as an invitation to all its neighbors. And its neighbors become its children in T.
3: **while** there are nodes outside of T **do** ▷ Takes $O(\log \Delta)$ rounds.
4: Each node $v \in T$ broadcasts its ID to the nodes in Ψ_v.
5: **if** node $u \notin T$ receives IDs from nodes in tree T **then**
6: u accept invitation based on the highest priority node, say v, and becomes v's child in T.
7: **end if**
8: **end while**

Lemma 5.6. *In $O(\log \Delta)$ rounds, all nodes are guaranteed to be part of the tree T.*

Proof. This is guaranteed from the use of leader election algorithm. Consider the graph G' constructed as follows. Let ℓ's neighbors be its neighbors in G. For every other node $v \neq \ell$ its neighbors are ψ_v. Clearly, from Algorithm 11, G' is connected (as every node learns of ℓ) and of diameter $O(\log \Delta)$. Let level i denote all nodes that are at most i hops away from ℓ in G'. We claim that in i rounds, all nodes in level i would become a part of the tree T. By using induction, this is clearly true for $i = 1$. Assuming it is true for i, nodes of $i + 1$ would become part of the tree next as they are in the Ψ_v of at least one node in level i and thus would get an invitation. And since the number of levels can be at most $O(\log \Delta)$, all nodes become part of T in $O(\log \Delta)$ rounds.

□

Lemma 5.7. *Algorithm 12 forms a tree of height $O(\log \Delta)$.*

Proof. Since in each iteration of the while loop, the height of the tree is extended by at most 1 (that is, by attaching children to the leaves of T). And since the algorithm ensures that all nodes have become a part of T in $O(\log \Delta)$ iterations of the while loop, the height of T can not be more than $O(\log \Delta)$. Notice that since each node accepts only one invite, there can be no creation of a cycle.

□

Remark: 5. *The diameter of the graph created by Algorithm 12 is* $O(\log \Delta)$.

Lemma 5.8. *Algorithm 12 takes* $O(n \log \Delta)$ *messages.*

Proof. In Algorithm 11, for every node v communication takes place via Ψ_v edges in $O(n \log \Delta)$ messages (Theorem 5.1). In Algorithm 12 (from Line 2 to Line 8) communication also takes place via same edges (Ψ_v) for two times. Therefore, message complexity remains unchanged to $O(n \log \Delta)$. □

Thus, from the above discussion, we conclude the following result.

Theorem 5.2. *There exists an algorithm that solves the broadcast problem in* $O(n \log \Delta)$ *messages and* $O(\log \Delta)$ *rounds which generate a tree of height* $O(\log \Delta)$.

5.5 Conclusion

We studied the leader election problem in diameter-two networks. We settled all the questions raised by Chatterjee et al. [31] w.r.t. deterministic setting. Various open problems come to light due to our work. These are as follows:

1. We presented an $O(\log \Delta)$-round and $O(n \log \Delta)$-message complexity algorithm for the explicit leader election. An interesting question is to reduce the round complexity to $O(1)$ while keeping the message complexity $O(n \log n)$?

2. Tree formed by broadcast has height $O(\log \Delta)$. An interesting question rises whether this is optimal when the message and round complexity remain unchanged or constant height is possible.

3. Is it possible to have a randomized algorithm (with high probability) with message complexity $O(n \log n)$ and constant round complexity without the knowledge of n?

4. Agreement is a weaker problem as compared to leader election, in which, each node has an input value and reaches on agreement w.r.t. one given input value. It would be interesting to see whether message complexity $\Omega(n \log n)$ is a tight bound w.r.t. agreement problem.

Chapter 6

Conclusion and Future Work

In this thesis, we developed fast algorithms for various fundamental distributed network problems including leader election, agreement, and graph realization problems in different settings. We studied the optimal round and message complexity algorithm and addressed some open questions raised in the other researchers' work. In some cases, our work also supported the theoretical analysis with experimental evaluation to highlight the effectiveness and efficiency of the work.

We presented the randomized (Monte Carlo) algorithms for the leader election and agreement problems in a crash-fault setting and also showed non-trivial lower bounds on the message complexity. Our algorithms achieved sublinear message complexity in the so-called implicit version of the two problems when tolerating more than a constant fraction of the faulty nodes. Specifically, our algorithms even work up to the $\mathrm{polylog}\, n$ number of non-faulty nodes. The message complexity (and also the time complexity) of our algorithms is optimal (up to a $\mathrm{polylog}\, n$ factor). Further, our algorithm works in anonymous networks, where nodes do not know each other. These are the first sub-linear results for both the leader election and the agreement problem in the crash-fault distributed networks. Then we studied the Byzantine agreement (BA) problem and focused on the implicit BA and provided a sublinear algorithm with matching lower bound up to $\mathrm{polylog}\, n$ factor in the presence of global coin and cryptographic assumptions. This is the first sublinear message complexity result of BA. A quadratic message lower bound is known for any deterministic BA protocol due to Dolev-Reischuk [39]. The existing randomized BA protocols had at least quadratic message complexity in the honest majority setting. Our results showed the power of a global coin in achieving significant improvement over the existing results. It can be viewed as a step towards understanding the message complexity of randomized BA in distributed networks with PKI and global coin.

Further, we discussed the graph realization problem in the Congested Clique with faults and provided efficient algorithms for realizing overlays for a given degree sequence. For that, we provided two models, namely, KT_1 and KT_0 based on the knowledge of the neighborhood. Our algorithms (in both the models) are simultaneously optimal in both the round and the message complexity. Towards the hindmost chapter of the thesis, we studied the leader election problem in diameter-two networks and settled all the questions raised by Chatterjee et al. [31] with respect to a deterministic setting. We affirmatively answered the question left open in their paper – whether

it is possible to remove the assumption on the knowledge of n. Another one is whether *explicit* leader election can be solved in $\tilde{O}(n)$ messages deterministically.

The thesis emphasizes the usefulness of the leader election and fault-tolerant computation. We are hopeful that the discussed methods and techniques would be helpful for future research works. Further, we believe that the work discussed in the thesis would open up some interesting research problems. We discussed several open problems raised by work at the end of each chapter. We like to highlight here some of them.

Recall that the thesis successfully achieved a sublinear message complexity algorithm for leader election and agreement in the static graph, specifically addressing the crash-fault and Byzantine scenarios in Chapter 2 and 3, respectively. Extensive research has recently been conducted on the static graph and general graph without any faults. In light of this, it becomes intriguing to explore the extension of the discussed problem by relaxing certain conditions. Is it possible, for example, to achieve a sublinear message complexity Byzantine agreement algorithm in this setting without utilizing a global coin or hash function? Additionally, another compelling question arises: Can a sublinear message bound be achieved when facing an adaptive adversary which can take control of Byzantine nodes at any point during the algorithm's execution? Furthermore, it is worth mentioning other problems of broader significance that involve more robust graph settings, such as dynamic graphs and general graphs in adversarial scenarios. These areas offer promising avenues for further exploration and investigation.

In the graph realization problem, we discussed the crash-fault setting. Some intriguing problems involve extending and defining the problem for both omission failure and Byzantine settings, with or without prior knowledge of the neighbors. These variations present interesting challenges and opportunities for exploration and analysis. These problems can be studied with respect to their round and message complexity, as well as towards the lower bounds. Finally, one can extend the diameter-two networks to the general setting of the networks and study the networks more extensively for the leader election problem.

In the long run, these ideas may be useful for a better understanding of the distributed network algorithm. Specifically, to break the symmetry and to deal with faulty networks robustly. Eventually, it would be helpful to handle failures of specific parts or nodes without impairing the performance or availability of the entire system. The set-up would be helpful to deal with practical problems like cluster computing[1] which can be used in big data processing. As the amount of data being generated grows exponentially, cluster computing can be used to process and analyze this data more efficiently. Another contribution may be in the emerging fields like artificial intelligence

[1]Cluster computing involves multiple interconnected computers to work together as a single system.

and machine learning which require large amounts of computing power to train models and process data, cluster computing can be used to speed up these processes and improve accuracy. Similarly, in the field of cryptocurrency, Bitcoin miners can use grid computing[2] to connect their computing resources with other miners worldwide to increase the chances of earning rewards.

[2]Grid computing uses geographically distributed resources to work together as a single system.

Bibliography

[1] Ittai Abraham, Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Symposium on Principles of Distributed Computing (PODC)*, pages 317–326, 2019.

[2] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected O(1) rounds, expected o(n^2) communication, and optimal resilience. In *Financial Cryptography and Data Security (FC)*, pages 320–334, 2019.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *Conference on Principles of Distributed Systems, (OPODIS)*, pages 25:1–25:19, 2017.

[4] Yehuda Afek and Eli Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. *SIAM Journal on Computing (SICOMP)*, 20(2):376–394, 1991.

[5] Adnan Agbaria and Roy Friedman. Overcoming byzantine failures using checkpointing. *University of Illinois at Urbana-Champaign Coordinated Science Laboratory technical report no. UILU-ENG- 03-2228 (CRHC-03-14)*, 2003.

[6] Dan Alistarh, James Aspnes, Valerie King, and Jared Saia. Communication-efficient randomized consensus. *Distributed Computing (DC)*, 31(6):489–501, 2018.

[7] Yair Amir, Claudiu Danilov, Jonathan Kirsch, John Lane, Danny Dolev, Cristina Nita-Rotaru, Josh Olsen, and David John Zage. Scaling byzantine fault-tolerant replication to wide area networks. In *International Conference on Dependable Systems and Networks (DSN)*, pages 105–114, 2006.

[8] David P. Anderson and John Kubiatowicz. The worldwide computer. *Scientific American*, 286(3):28–35, 2002.

[9] Srinivasa R. Arikati and Anil Maheshwari. Realizing degree sequences in parallel. *SIAM Journal on Discrete Mathematics (SIDMA)*, 9(2):317–338, 1996.

[10] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms (TALg)*, 3(4):37–es, 2007.

[11] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, 2004.

[12] John Augustine, Keerti Choudhary, Avi Cohen, David Peleg, Sumathi Sivasubramaniam, and Suman Sourav. Distributed graph realizations. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 33(6):1321–1337, 2022.

[13] John Augustine, Mohsen Ghaffari, Robert Gmyr, Kristian Hinnenthal, Christian Scheideler, Fabian Kuhn, and Jason Li. Distributed computation in node-capacitated networks. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 69–79, 2019.

[14] John Augustine, Anisur Rahaman Molla, and Gopal Pandurangan. Sublinear message bounds for randomized agreement. In *Symposium on Principles of Distributed Computing (PODC)*, pages 315–324, 2018.

[15] John Augustine and Sumathi Sivasubramaniam. Spartan: A Framework For Sparse Robust Addressable Networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1060–1069, 2018.

[16] Amitabha Bagchi, Ankur Bhargava, Amitabh Chaudhary, David Eppstein, and Christian Scheideler. The effect of faults on network expansion. *Theory of Computing Systems (TCS)*, 39(6):903–928, 2006.

[17] Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *Symposium on Principles of Distributed Computing (PODC)*, page 193–199, 1998.

[18] Amotz Bar-Noy, Toni Böhnlein, David Peleg, Mor Perry, and Dror Rawitz. Relaxed and approximate graph realizations. In *International Workshop on Combinatorial Algorithms (IWOCA)*, pages 3–19, 2021.

[19] Amotz Bar-Noy, Toni Böhnlein, David Peleg, and Dror Rawitz. On vertex-weighted graph realizations. In *International Conference on Algorithms and Complexity (CIAC)*, pages 90–102, 2021.

[20] Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Graph realization of distance sets. In *International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 13:1–13:14, 2022.

[21] Leonid Barenboim and Victor Khazanov. Distributed symmetry-breaking algorithms for congested cliques. In *International Computer Science Symposium in Russia*, pages 41–52, 2018.

[22] Florent Becker, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. The impact of locality on the detection of cycles in the broadcast congested clique model. In *Latin American Symposium (LATIN)*, pages 134–145, 2018.

[23] Mehdi Behzad and James E Simpson. Eccentric sequences and eccentric sets in graphs. *Discrete Mathematics*, 16(3):187–193, 1976.

[24] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.

[25] Michael Ben-Or, Elan Pavlov, and Vinod Vaikuntanathan. Byzantine agreement in the full-information model in o(log n) rounds. In *Symposium on Theory of Computing (STOC)*, pages 179–186, 2006.

[26] Bitcoin. Bitcoin website https://bitcoin.org/.

[27] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *Symposium on Principles of Distributed Computing (PODC)*, pages 57–64, 2013.

[28] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[29] Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing (DC)*, 34(6):463–487, 2021.

[30] Tushar Deepak Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In *PODC2007*, pages 398–407. ACM, 2007.

[31] Soumyottam Chatterjee, Gopal Pandurangan, and Peter Robinson. The complexity of leader election in diameter-two networks. *Distributed Comput.*, 33(2):189–205, 2020.

[32] Bogdan S. Chlebus and Dariusz R. Kowalski. Gossiping to reach consensus. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–229, 2002.

[33] Bogdan S. Chlebus and Dariusz R. Kowalski. Robust gossiping with an application to consensus. *Journal of Computer and System Sciences (JCSS)*, pages 1262–1281, 2006.

[34] Bogdan S. Chlebus and Dariusz R. Kowalski. Locally scalable randomized consensus for synchronous crash failures. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 290–299, 2009.

[35] Bogdan S. Chlebus, Dariusz R. Kowalski, and Jan Olkowski. Brief announcement: Deterministic consensus and checkpointing with crashes: Time and communication efficiency. In *Symposium on Principles of Distributed Computing (PODC)*, pages 106–108, 2022.

[36] Bogdan S. Chlebus, Dariusz R. Kowalski, and Michal Strojnowski. Fast scalable deterministic consensus for crash failures. In *Symposium on Principles of Distributed Computing (PODC)*, pages 111–120, 2009.

[37] Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *Journal of the ACM (JACM)*, pages 591–614, 1989.

[38] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting. In *International Symposium on Distributed Computing*, pages 195–209. Springer, 2012.

[39] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[40] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[41] Danny Dolev and Raymond Strong. Requirements for agreement in a distributed system. In *International Symposium on Distributed Data Bases*, pages 115–129, 1982.

[42] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Symposium on Principles of distributed computing (PODC)*, pages 367–376, 2014.

[43] Paul Erdös and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.

[44] Ethereum. Ethereum website https://ethereum.org/.

[45] Paul Feldman and Silvio Micali. Byzantine agreement in constant expected time (and trusting no one). In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 267–276, 1985.

[46] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing (SICOMP)*, 26(4):873–933, 1997.

[47] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002.

[48] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters (IPL)*, 14(4):183–186, 1982.

[49] Matthias Fitzi. Generalized communication and security models in byzantine agreement. *PhD Dissertation*, 2002.

[50] András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics (SIDMA)*, 5:25–43, 1992.

[51] András Frank. Connectivity augmentation problems in network design. In *Mathematical Programming: State of the Art*, pages 34–63. Univ. Michigan, 1994.

[52] Howard Frank and Wushow Chou. Connectivity considerations in the design of survivable networks. *IEEE Transactions on Circuit Theory (TCT)*, CT-17:486–490, 1970.

[53] Zvi Galil, Alain J. Mayer, and Moti Yung. Resolving message complexity of byzantine agreement and beyond. In *Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, 1995.

[54] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1):66–77, 1983.

[55] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM Journal on Computing (SICOMP)*, 27(1):247–290, 1998.

[56] Mohsen Ghaffari and Krzysztof Nowicki. Congested Clique Algorithms for the Minimum Cut Problem. In *Symposium on Principles of Distributed Computing (PODC)*, pages 357–366, 2018.

[57] Mohsen Ghaffari and Merav Parter. MST in log-star rounds of congested clique. In *Symposium on Principles of Distributed Computing (PODC)*, pages 19–28, 2016.

[58] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[59] Seth Gilbert and Dariusz R. Kowalski. Distributed agreement with optimal communication complexity. In *Symposium on Discrete Algorithms (SODA)*, pages 965–977, 2010.

[60] Seth Gilbert, Peter Robinson, and Suman Sourav. Leader election in well-connected graphs. In *Symposium on Principles of Distributed Computing (PODC)*, pages 227–236, 2018.

[61] Shafi Goldwasser, Elan Pavlov, and Vinod Vaikuntanathan. Fault-tolerant distributed computing in full-information networks. In *Symposium on Foundations of Computer Science (FOCS)*, pages 15–26, 2006.

[62] Ralph E. Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics (JSTOR)*, 9, 1961.

[63] Jim Gray. The cost of messages. In *Symposium on Principles of Distributed Computing (PODC)*, pages 1–7, 1988.

[64] Diksha Gupta, Jared Saia, and Maxwell Young. Resource burning for permissionless systems. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 19–44, 2020.

[65] Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement. *Mathematical Systems Theory*, 26(1):41–102, 1993. Conference version in Symposium on Principles of Distributed Computing (PODC) 1991.

[66] Mohammad Taghi Hajiaghayi, Dariusz R. Kowalski, and Jan Olkowski. Improved communication complexity of fault-tolerant consensus. In *Symposium on Theory of Computing (STOC)*, pages 488–501, 2022.

[67] Seifollah Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph - i. *Journal of the Society for Industrial and Applied Mathematics (JSTOR)*, 10(3):496–506, 1962.

[68] Václav Havel. A remark on the existence of finite graphs. *Časopis pro pěstování matematiky*, 80:477–480, 1955.

[69] Pierre A. Humblet. Electing a leader in a clique in o(n log n) messages. In *IEEE Conference on Decision and Control (CDC)*, pages 1139–1140, 1984.

[70] Tomasz Jurdzinski and Krzysztof Nowicki. Connectivity and minimum cut approximation in the broadcast congested clique. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO), Revised Selected Papers*, pages 331–344, 2018.

[71] Tomasz Jurdziński and Krzysztof Nowicki. MST in $o(1)$ rounds of congested clique. In *Symposium on Discrete Algorithms (SODA)*, pages 2620–2632, 2018.

[72] Bruce Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. In *Symposium on Discrete Algorithms (SODA)*, pages 1038–1047, 2008.

[73] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences (JCSS)*, 75(2):91–112, 2009.

[74] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing (DC)*, 25(3):189–205, 2012.

[75] Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{o}(n^{3/2})$ bits. In *International Symposium on Distributed Computing (DISC)*, pages 464–478, 2009.

[76] Valerie King and Jared Saia. Scalable byzantine computation. *SIGACT News*, 41(3):89–104, 2010.

[77] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *Journal of the ACM (JACM)*, 58(4):18:1–18:24, 2011.

[78] Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *Journal of the ACM (JACM)*, 63(2):13:1–13:21, 2016.

[79] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Symposium on Discrete Algorithms (SODA)*, pages 990–999, 2006.

[80] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via

collective signing. In *USENIX Security Symposium (USENIX Security)*, pages 279–296, 2016.

[81] Jiejun Kong. *Anonymous and untraceable communications in mobile wireless networks*. University of California, Los Angeles, 2004.

[82] Christian Konrad. Mis in the congested clique model in $\log \log \Delta$ rounds. *arXiv preprint arXiv:1802.07647*, 2018.

[83] Ephraim Korach, Shay Kutten, and Shlomo Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):84–101, 1990.

[84] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *Symposium on Principles of Distributed Computing (PODC)*, pages 199–207, 1984.

[85] 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. *SIAM Journal on Computing (SICOMP)*, 16(2):231–236, 1987.

[86] Ephraim Korach, Shlomo Moran, and Shmuel Zaks. Optimal lower bounds for some distributed algorithms for a complete network of processors. *Theory of Computing Systems (TCS)*, 64(1):125–132, 1989.

[87] Janne H. Korhonen and Jukka Suomela. Towards a complexity theory for the congested clique. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 163–172, 2018.

[88] Dariusz R. Kowalski and Jaroslaw Mirek. On the complexity of fault-tolerant consensus. In *Networked Systems International Conference (NETYS), Revised Selected Papers*, pages 19–31, 2019.

[89] Dariusz R. Kowalski and Achour Mostéfaoui. Synchronous byzantine agreement with nearly a cubic number of communication bits. In *Symposium on Principles of Distributed Computing (PODC)*, pages 84–91, 2013.

[90] Dariusz R. Kowalski and Miguel A. Mosteiro. Time and communication complexity of leader election in anonymous networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 449–460, 2021.

[91] Manish Kumar. Fault-tolerant graph realizations in the congested clique, revisited. In *International Conference on Distributed Computing and Intelligent Technology (ICDCIT)*, pages 84–97, 2023.

[92] Manish Kumar and Anisur Rahaman Molla. On the message complexity of fault-tolerant computation: Leader election and agreement. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 34(4):1115–1127, 2023.

[93] Manish Kumar and Anisur Rahaman Molla. Sublinear message bounds of authenticated implicit byzantine agreement. In *International Conference on Distributed Computing and Networking – to be published (ICDCN)*, 2024.

[94] Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Fault-tolerant graph realizations in the congested clique. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 108–122, 2022.

[95] Manish Kumar, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Improved deterministic leader election in diameter-two networks. In *International Conference on Algorithms and Complexity (CIAC)*, pages 323–335, 2023.

[96] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. On the complexity of universal leader election. *Journal of the ACM (JACM)*, 62(1):7:1–7:27, 2015.

[97] Shay Kutten, Gopal Pandurangan, David Peleg, Peter Robinson, and Amitabh Trehan. Sublinear bounds for randomized leader election. *Theoretical computer science (TCS)*, 561:134–143, 2015.

[98] Shay Kutten, Peter Robinson, Ming Ming Tan, and Xianbin Zhu. Improved tradeoffs for leader election. In *Symposium on Principles of Distributed Computing (PODC)*, pages 355–365, 2023.

[99] Shay Kutten and Dmitry Zinenko. Low communication self-stabilization through randomization. In *International Symposium on Distributed Computing (DISC)*, pages 465–479, 2010.

[100] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[101] Gérard Le Lann. Distributed systems - towards a formal approach. In *International Federation for Information Processing (IFIP)*, pages 155–160, 1977.

[102] Linda Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6(4):287–293, 1975.

[103] Na Li, Nan Zhang, Sajal K Das, and Bhavani Thuraisingham. Privacy preservation in wireless sensor networks: A state-of-the-art survey. *Ad Hoc Networks*, 7(8):1501–1514, 2009.

[104] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $o(\log \log n)$ communication rounds. *SIAM Journal on Computing (SICOMP)*, 35(1):120–131, 2005.

[105] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ritu Sharma, and Sharon Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials*, 7(2):72–93, 2005.

[106] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[107] Apostolos Malatras. State-of-the-art survey on P2P overlay networks in pervasive computing environments. *International Journal of Network and Computer Applications (IJNCA)*, 55:1–23, 2015.

[108] Dahlia Malkhi and Michael K. Reiter. Unreliable intrusion detection in distributed computations. In *Computer Security Foundations Workshop (CSFW)*, pages 116–125, 1997.

[109] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[110] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *Symposium on Foundations of Computer Science (FOCS)*, pages 120–130, 1999.

[111] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. https://dspace.mit.edu/handle/1721.1/107927, 2017.

[112] Michael Mitzenmacher and Eli Upfal. Probability and computing: Randomized algorithms and probabilistic analysis. 2004.

[113] Atsuki Momose and Ling Ren. Optimal communication complexity of byzantine consensus under honest majority. *Journal of Environmental Sciences (China) English Ed*, 2020.

[114] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[115] Boaz Patt-Shamir and Marat Teplitsky. The round complexity of distributed sorting. In *Symposium on Principles of Distributed Computing (PODC)*, pages 249–256, 2011.

[116] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[117] David Peleg. Time-optimal leader election in general networks. *Journal of Parallel and Distributed Computing (JPDC)*, 8(1):96–99, 1990.

[118] David Peleg. Distributed computing: A locality-sensitive approach. 2000.

[119] Michael O. Rabin. Randomized byzantine generals. In *Symposium on Foundations of Computer Science (FOCS)*, pages 403–409, 1983.

[120] Mohsin Ur Rahman. Leader election in the internet of things: Challenges and opportunities. *CoRR*, abs/1911.00759, 2019.

[121] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *Special Interest Group on Data Communication (SIGCOMM)*, pages 161–172, 2001.

[122] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.

[123] Mohammed Refai, Ahmad Abdel-Aziz Sharieh, and Fahad Alshammari. Leader election algorithm in 2d torus networks with the presence of one link failure. *International Arab Journal of Information Technology (IAJIT)*, 7(2):105–114, 2010.

[124] Sean C. Rhea, Patrick R. Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y. Zhao, and John Kubiatowicz. Pond: The oceanstore prototype. In *USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[125] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms (IFIP)*, pages 329–350, 2001.

[126] Nicola Santoro. *Design and analysis of distributed algorithms*. Wiley series on parallel and distributed computing. Wiley, 2007.

[127] Christian Scheideler. Models and techniques for communication in dynamic networks. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 27–49, 2002.

[128] Elaine Shi and Adrian Perrig. Designing secure sensor networks. *IEEE Wireless Communications*, 11(6):38–43, 2004.

[129] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer networks*, 76:146–164, 2015.

[130] Gurdip Singh. Efficient distributed algorithms for leader election in complete networks. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 472–479, 1991.

[131] 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. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[132] Gerard Tel. Introduction to distributed algorithms. *Cambridge University Press*, 1994.

[133] Eli Upfal. Tolerating linear number of faults in networks of bounded degree. In *Symposium on Principles of Distributed Computing (PODC)*, pages 83–89, 1992.

[134] Rolf H Weber. Internet of things–new security and privacy challenges. *Computer law & security review (CLSR)*, 26(1):23–30, 2010.

[135] Alex Wright. Contemporary approaches to fault tolerance. *Communications of the ACM*, 52(7):13–15, 2009.

[136] Assaf Yifrach and Yishay Mansour. Fair leader election for rational agents in asynchronous rings and networks. In *Symposium on Principles of Distributed Computing (PODC)*, pages 217–226, 2018.

[137] Hiroyuki Yoshino, Naohiro Hayashibara, Tomoya Enokido, and Makoto Takizawa. Byzantine agreement protocol using hierarchical groups. In *International Conference on Parallel and Distributed Systems (ICPADS)*, pages 64–70, 2005.