© 2012 Guanfeng Liang

NETWORK-AWARE MECHANISMS FOR TOLERATING BYZANTINE FAILURES IN DISTRIBUTED SYSTEMS

BY

GUANFENG LIANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Nitin H. Vaidya, Chair
Associate Professor Chandra Chekuri
Adjunct Professor P.R. Kumar
Professor Venugopal V. Veeravalli

ABSTRACT

Given the growing reliance of industry and government on online information services such as cloud computing and data centers, efficient fault-tolerance algorithm design is of increasing importance in both industry and academia. In this dissertation, we present some of our efficient fault-tolerant algorithms for distributed systems under both point-to-point and broadcast communication models.

For the point-to-point model, we mainly consider Byzantine agreement algorithms. We develop algorithms that require only $O(nL)$ total bits of communication for achieving agreement of L bits among n nodes for sufficiently large L, without making any cryptographic assumption. Previous algorithms either have higher communication cost or rely on cryptographic assumptions. We also develop Byzantine agreement algorithms that perform well when the communication links in the network are capacity-constrained. We develop the first Byzantine broadcast algorithm that achieves constant fraction of the optimal throughput in general point-to-point networks. For some special class of networks, we develop algorithms that achieve the optimal throughput. We then study the communication complexity of the multiparty equality function, which is the core of the Byzantine agreement problem.

For the broadcast model, we study the problem of detecting packet tampering attacks in multi-hop wireless networks. We propose a lightweight detection scheme that integrates the idea of wireless watchdogs and error detection coding. We show in a single flow example that even if the watchdog can only observe a fraction of packets, by choosing the encoder properly, an attacker will be detected with high probability while achieving throughput arbitrarily close to optimal. The trade-off between throughput and security in a more practical setting – there are multiple data flows in the network and a distributed random access MAC protocol is used – is also studied.

To my wife and parents, for their love and support

ACKNOWLEDGMENTS

My deepest gratitude is to my advisor, Prof. Nitin Vaidya, for his guidance, understanding, patience, and most importantly, his friendship during my graduate studies at the University of Illinois at Urbana-Champaign. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. He taught me how to question thoughts and express ideas. His patience and support helped me overcome many crisis situations and finish this dissertation.

I thank Prof. Chekuri, Prof. Kumar and Prof. Veeravalli for serving on my doctoral committee, and for their insightful comments. Many past and current colleagues in the Distributed Algorithms and Wireless Networking Group have been very helpful in many ways during my graduate school days; I thank all of them, in particular Ghazale Hosseinabadi, Tae Hyun Kim, Vijay Raman, Shehla Saleem Rana, Benjamin Sommer and Lewis Tseng. Special thanks to Benjamin Sommer, who has contributed significantly in implementation of some of the proposed algorithms and experimental evaluation, as elaborated in Sections 2.7.4 and 2.7.5.

I acknowledge the National Science Foundation and the U.S. Army Research Office for financially supporting this research.

Finally, and most importantly, I would like to thank my wife Ying Ding. Her support, encouragement, quiet patience and unwavering love are undeniably the bedrock upon which the past eight years of my life have been built. Her tolerance of my occasional vulgar moods is a testament in itself of her unyielding devotion and love. I thank my parents, Yonghong Zhang and Jinpei Liang, for their faith in me and allowing me to pursue the life that I wanted. Their support during these years made this dissertation possible.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

BA	Byzantine Agreement
BFT	Byzantine Fault-Tolerance
MEQ	Multiparty Equality
CBC	Coding-Based Byzantine Consensus
CBB	Coding-Based Byzantine Broadcast
NAB	Network-Aware Byzantine Broadcast
CAB	Capacity-Achieving Network-Aware Byzantine Broadcast

CHAPTER 1

INTRODUCTION

In recent years, we have seen tremendous growth in the development in distributed computing systems, which consist of a multitude of autonomous computing devices that communicate through a network, wired or wireless. In such systems, applications often require cooperation between multiple devices, which are usually failure-prone. In many systems, such failures can lead to unanticipated, potentially disruptive failure behavior and to service unavailability. Hence, *fault-tolerance* is of increasing importance in the design of such distributed network systems.

To achieve fault-tolerance in a distributed system, one must incorporate redundancy, in the forms of information and/or processing components. Intuitively, the more redundancy introduced, the more fault-tolerant the system can be made. However, more redundancy usually also means a less efficient system. The goal of this dissertation is to investigate the trade-off between the amount of redundancy required and the level of fault-tolerance in distributed systems under both point-to-point and broadcast communication models. For the point-to-point model, we will mainly consider the Byzantine fault-tolerance problem. For the broadcast model, we will focus on the problem of detecting packet tampering attack by faulty intermediate nodes in wireless multi-hop networks.

1.1 Research Summary

1.1.1 Communication Complexity of Byzantine Fault-Tolerance

Recent years have seen a tremendous growth in the popularity of data-oriented online services. Enterprises often run their critical business ap-

plications using "clouds" that provide storage as well as computing services. Many individual users also have become dependent on the Internet to store their personal data including photos, music and videos. As industry and individuals rely increasingly on data centers and other similar online services, the threat posed by malicious attacks and software errors has also become increasingly prominent. For example, software errors have brought down the Amazon S3 storage system for several hours [1], and have resulted in mass email deletion at Gmail [2]. Being able to provide reliable and consistent access to the data and services has become an important quality-of-service requirement that the online services must fulfill.

Byzantine fault-tolerance (BFT) provides a powerful state machine replication approach for building highly reliable services despite failures (or attacks). In BFT state machine replication, n replicas collectively behave as one *fault-free* server, even if up to f replicas, with $f < n/3$, are faulty and deviate from the protocol, i.e., *misbehave*, in arbitrary (or "Byzantine") fashion [3]. A key requirement in implementing BFT is that all the fault-free replicas of a server must execute identical client requests in an agreed upon order. A *Byzantine agreement* algorithm is used to reach an agreement on the requests to be handled. The agreement on requests guarantees that all fault-free replicas always have consistent state and produce the same output. Then the clients can obtain the correct output from the replicated server by taking a majority vote of the individual replica outputs. The Byzantine fault model allows for worst-case behavior by the faulty nodes. The fault model can be used to characterize the behavior of a replica under attack (security compromise), or to model the detrimental impact of undetected failures or software bugs.

Despite its potential for achieving a high level of reliability in the presence of *arbitrary* nature of failure or attack, BFT was rarely adopted in practice in the three decades after its introduction in 1980 [3]. The reluctance to use BFT in practical systems stems from two causes: (a) the high overhead of Byzantine agreement, and (b) the belief that the Byzantine fault model is too pessimistic to model real failures that occur in practical systems.

From the high profile failures in recent years (e.g., [1, 2]), it is apparent that not all failures in practical systems are "crash" or "fail-stop" failures, and at least for some critical services, higher level of reliability offered by BFT is desired. Thus, provided that the overheads are reduced, BFT can become attractive for some applications.

In 1985, Dolev and Reischuk [4] proved that, without authentication, $\Theta(n^2)$ bits are necessary to be communicated, in order to achieve agreement on 1 bit, which results in a lower bound on the *per-bit communication complexity* of agreement as $\Omega(n^2)$. This result is one of the main reasons why BFT is perceived as being too expensive to be practical.

The objective of this part of the dissertation is to mitigate the *communication overhead* of BFT. We show that, if the size of the value/message to be agreed on is sufficiently large, the per-bit communication complexity for agreement can be made much lower than $\Theta(n^2)$. In particular, we design and analyze algorithms that achieve Byzantine agreement of value/message of L bits among n nodes with at most $f < n/3$ faulty nodes, while requiring $O(nL)$ bits of communication, even in the worst case. In addition to developing theoretical algorithms, we also demonstrate and evaluate them via experimental implementations.

Work reported in this part has resulted in conference publications [5, 6] and technical reports [7, 8].

1.1.2 Network-Aware Byzantine Fault-Tolerance

The past work on BFT has not considered processing or communication bottlenecks when designing the Byzantine agreement algorithms. In particular, previous algorithms do not consider the possibility that some of the communication links in the network may be constrained. These algorithms treat all communication links in the network *equally*, and usually allocate roughly the same amount of workload to each communication link in the network. However, in practice, different links in the network may have very different capacities. When applying the previous BFT algorithms in such systems, the links with poor capacity can become a bottleneck: transmission of the workload over the bottleneck links takes a long time, so accordingly, completion of agreement will take longer.

The objective of this part of the dissertation is to improve performance of BFT algorithms by taking into account the communication constraints of the underlying network. We say such algorithms are "*network-aware*". We consider the performance metric of *throughput* of network-aware BFT algorithms. The notion of throughput here is similar to that used in the

networking/communications literature on unicast or multicast traffic. Informally, the throughput of a BFT algorithm in a given network is the average number of bits that can be agreed on per unit time, while the traffic imposed by the algorithm stays within the communication constraints of the networks.

In particular, we study the throughput of BFT algorithms in point-to-point networks, in which each communication link is subject to a capacity constraint. We derive an upper bound on the optimal throughput over all possible BFT algorithms in general point-to-point networks. We develop a network-aware BFT algorithm that is guaranteed to achieve at least 1/3 of the optimal throughput in general point-to-point networks. It is the first BFT algorithm that achieves constant fraction of the optimal throughput in general networks. Moreover, for certain classes of point-to-point networks, we also develop BFT algorithms that achieve the optimal throughput.

Work reported in this part has resulted in conference publications [9, 10, 11] and technical reports [12, 13, 14, 15, 16].

1.1.3 Multiparty Equality Function Computation under Point-to-Point Communication Model

In this part, we study the communication complexity of distributed computation of the multiparty equality function (MEQ), under the point-to-point communication model. We generalize the two-party communication complexity model [17] to the multiparty scenario: $n \geq 2$ nodes are trying to compute a function $f(x_1, \cdots, x_n)$ by communicating over point-to-point links, with input $x_i \in X_i$ only known to node i.

We demonstrate that traditional techniques generalized from the two-party communication complexity problem are not sufficient to obtain tight bounds under the point-to-point communication model. We then introduce techniques to significantly reduce the space of protocols to study. We then study the MEQ problem of $n = 3$, $|X_i| = 6$, and introduce a protocol that achieves the optimal complexity. This protocol is then used as a building block for construction of efficient protocols for large $|X_i|$. The problem of finding the communication complexity of the MEQ problem for general values of n and $|X_i|$ is still open.

Work reported in this part has resulted in a conference publication [18]

and a technical report [19].

1.1.4 Watchdog in Wireless Networks

In wireless ad hoc and sensor networks, paths between a source and destination are usually multihop, and data packets are relayed in several wireless hops from their source to their destination. This multihop nature makes the wireless networks subject to tampering attack: a compromised/misbehaving node can easily ruin data communications by dropping or corrupting packets it should forward. Watchdog mechanism [20] is a monitoring method used for such networks. The basic idea is to have nodes (called watchdogs) to monitor their neighborhoods using overheard messages in order to detect misbehavior, utilizing the broadcast nature of the wireless medium.

The main challenge for the watchdog mechanism is to balance the system throughput and ability to detect *most* tampering attacks, given the unreliability of the wireless environment. In this problem, we propose a computationally simple scheme that integrates source error detection coding and the watchdog mechanism. We show that by choosing the encoder properly, a misbehaving node will be detected with high probability while the throughput approaches optimal, even in the case when the watchdog can only overhear a fraction of the packets and the attacker is omniscient, i.e., knows what encoder is being used and no secret is shared only between the source and destination.

Work reported in this part has resulted in a conference publication [21] and a technical report [22].

1.2 Dissertation Outline

- In Chapter 2, we present our Byzantine fault-tolerance algorithms that achieve linear per-bit complexity, for sufficiently large input size. Proofs for correctness and experimental comparison with existing algorithms are also provided.

- In Chapter 3, we motivate the problem of designing network-aware BFT algorithms under capacity constraints of the underlying commu-

nication network. We first present one BFT algorithm that is guaranteed to achieve a constant fraction of the optimal throughput in general point-to-point networks. Then two algorithms that achieve optimal throughput in two special classes of point-to-point networks are presented.

- In Chapter 4, we study the distributed computation of the multiparty equality function.
- In Chapter 5, we introduce the idea of the integration of error-detection coding and the watchdog mechanism. We then provide analytical results on some case studies.
- In Chapter 6, we summarize the list of future work.
- Finally, in Chapter 7 we conclude the dissertation.

CHAPTER 2

ERROR-FREE BYZANTINE FAULT TOLERANCE WITH LINEAR COMPLEXITY

2.1 Introduction

In recent years, there has been a tremendous growth in the popularity of data-oriented online services, such as cloud computing, data centers and online storage. For example, industrial leaders such as Amazon, Google, IBM and Microsoft have been investing heavily in developing their cloud computing systems and data centers. As the reliance of industry, government, and individuals on data centers and other similar online information services increases, so does the threat posed by malicious attacks and software errors [1, 2]. Consequently, being able to provide reliable and consistent access to the data and services that they host has become an important requirement that these online services must fulfill.

Byzantine fault-tolerance (BFT), also known as Byzantine agreement (BA) in the theoretical distributed computing literature, provides a powerful state machine replication approach for providing highly reliable and consistent services in spite of the presence of failures. In BFT state machine replication, $n \geq 3f + 1$ replicas collectively behave as one *fault-free* server, even if up to f replicas are faulty and deviate from the algorithm, i.e., *misbehave*, in arbitrary (Byzantine) fashions [3]. The key of BFT is to make sure that all replicas agree upon the same sets as well as the order of requests, and execute them in the agreed upon order. This guarantees that all fault-free replicas always have consistent states and produce the same output. Then the correct output can be obtained by taking the majority of the individual outputs, since at least $2n/3$ of the replicas are fault-free.

Unfortunately, BFT has been rarely adopted in practice for the three decades following its introduction in 1980 [3], mainly because of the overhead incurred due to the minimum $3f + 1$ replicas required to tolerate f failures,

and the high communication overhead of previously proposed *error-free* BFT algorithms.

In the last decade, there have been numerous efforts devoted to making BFT systems *practical* [23, 24, 25, 26, 27, 28]. One common optimization is that, to check the consistency of a request (or a piece of data) received by different replicas, the replicas exchange hash values (sometimes called *"digests"*) computed from the request, using some collision-resistant hash function, and check the hash values against the original request, instead of exchanging the entire request. Since the digest is much smaller than the original request in size, communication cost is significantly reduced (roughly by an order of n). Despite the impressive performance improvement achieved by these systems, the use of a collision-resistant hash function may, in fact, be problematic, for the following two reasons:

- First of all, the correctness of the aforementioned algorithms relies on the collision-resistant property of the hash function used. With the rapid improvement in modern cryptanalysis and computational power of computers, defeating the hash functions may become computationally feasible in the future, and a malicious adversary will be able to find collisions of the hash function and then break the system. For example, Castro and Liskov's seminal practical Byzantine fault tolerance (PBFT) [23] algorithm uses MD5, which has since been broken [29].

- The second reason is related to the first one. In a later implementation of PBFT in 2002 [25], MD5 was replaced by SHA-1 in order to improve the reliability of the algorithm. However, SHA-1 was then broken in 2008 [30]. It is likely that these algorithms need to use more and more secure hash functions (e.g. SHA-256 and SHA-512), trying to stay ahead of the development in technology and cryptanalysis. However, the more secure a hash function, the more expensive it is to compute. So the improvement we gain from reducing communication overhead with hashing can become overwhelmed by the increasing computational/time cost we pay for using a stronger hash function.

The discussion above motivates our work in this chapter. In particular, we ask the following question:

Is it possible to design a practical BFT system with the following two properties?

1. *Reliability: Always correct (in other words "error-free") and does not rely on hash functions; and*

2. *Efficiency: Performance is comparable to the aforementioned systems that use hash functions.*

We give an *affirmative* answer to this question in this chapter:

- We present CBC – a Coding-based Byzantine consensus algorithm, and CBB – a Coding-based Byzantine broadcast algorithm. These two algorithms solve two variants of the BFT problem: Consensus and Broadcast (formally defined later), respectively, without using any hash function or relying on any cryptographic assumption. Both algorithms are proved to be error-free, i.e., reliable.

- We prove that the communication complexity of both CBC and CBB are $O(nL)$ for sufficiently large input size L (formally defined later). Moreover, we show that

  - The communication complexity of CBC is at least one order of n lower than all previously known error-free Byzantine consensus algorithms. The best previously known communication complexity of error-free consensus algorithms is $O(n^2L)$.

  - The communication complexity of CBB is at most 1/2 of the best previously known communication complexity of error-free Byzantine broadcast algorithm, developed by Beerliova-Trubiniova and Hirt [31].

- Experimental results on our testbed show that the CBB algorithm is at least as efficient as the algorithms that use hash functions.

This chapter is structured as follows. We begin by giving formal problem formulation and describing our system and failure models. Related work is discussed in Section 2.3. Salient features of the two proposed algorithms are then briefly discussed in Section 2.4. Then details of CBC and CBB will be discussed in Section 2.5 and Section 2.7. Last, we summarize this chapter in Section 2.8

2.2 Problem Definitions and Models

We first introduce the formal definition of the two versions of the Byzantine agreement problem that we are going to study, as well as the system model that will be considered in the rest of this chapter.

Byzantine Consensus (BC)

The Byzantine consensus problem considers n nodes, namely nodes $1, ..., n$, of which at most f nodes may be *faulty* and deviate from the algorithm in arbitrary fashion. Each node i is given an L-bit input value x_i. The basic version of the consensus problem considered here requires that the following properties to be satisfied.

- **Termination**: every node i eventually decides on an output value y_i.
- **Consistency**: the output values of all fault-free nodes are equal, i.e., for every fault-free node i, $y_i = y$ for some y.
- **Validity**: if every node i holds the same input $x_i = x$ for some x, then $y = x$.

These properties have been used as the requirements for consensus in previous literature as well [28]. Other characterizations of the *validity* condition above are also of potential interest in practice. For instance, we may want the nodes to agree on the majority of the input values (if there is any), i.e., the agreed output value $y = x$ if at least $\lceil \frac{n+1}{2} \rceil$ nodes have input value equal to x. With suitable parameterization, the CBC algorithm satisfies such more general validity conditions as well (Section 2.5.5).

Byzantine Broadcast (BB)

Byzantine broadcast considers a similar problem. There are also n nodes $1, \cdots, n$ in the system. One special node is designated as the *source/sender*. Without loss of generality, assume that node n is the source. The other nodes $1, \cdots, n-1$ are designated as the *peers*. The source node 1 is given an L-bit input value x, which the source node tries to broadcast to the peers. The goal is for all the fault-free nodes to "agree on" the value being sent by

the source, despite the possibility that some of the nodes (possibly including the source) may be faulty. In particular, the following conditions must be satisfied:

- **Termination**: every fault-free peer i ($i < n$) eventually decides on an output value y_i.
- **Consistency**: the output values of all fault-free peers are equal, i.e., for every fault-free peer i ($i < n$), $y_i = y$ for some y.
- **Validity**: if the source is fault-free, then $y = x$.

A consensus (or broadcast) algorithm is said to be *error-free* if in all possible executions of the algorithm, the properties of BC (or BB) are always satisfied. We are interested in the communication complexity of error-free consensus and broadcast algorithms. *Communication complexity* of an algorithm is defined as the maximum (over all possible executions) of the total number of bits transmitted by all the nodes according to the specification of the algorithm. This measure of complexity was first introduced by Yao [17], and has been used widely (e.g., [32, 28, 33]).

System Model

We assume a synchronous fully connected network of n nodes. Every pair of nodes is connected by a pair of directed point-to-point communication channels. Each node correctly knows the identity of the nodes at the other end of its channels. Whenever a node receives a message on such a directed channel, it can correctly assume that the message is sent by the node at the other end of the channel. We assume a Byzantine adversary that has complete knowledge of the state of the nodes, including the L-bit input value(s). No secret is hidden from the adversary. The adversary can take over up to f nodes ($f < n/3$) at any point during the algorithm. These nodes are said to be *faulty*. The faulty nodes can engage in any "*misbehavior*", i.e., deviations from the algorithm, including collusion. The remaining nodes are *fault-free* and follow the algorithm.

2.3 Related Work

Binary agreement: Binary agreement corresponds to $L = 1$ in our notation. For binary agreement, optimal error-free algorithms (consensus and broadcast) with communication complexity $O(n^2)$ have been proposed [34, 35]. King and Saia [36] introduced a randomized *consensus* algorithm with communication complexity $O(n^{1.5})$, allowing a non-zero probability of error.

Multi-valued agreement: Fitzi and Hirt [28] proposed a multi-valued *consensus* algorithm in which an L-bit value (or message) is first reduced to a much shorter message, using a universal hash function. Byzantine consensus is then performed for the shorter hashed values. Given the result of consensus on the hashed values, consensus on L bits is then achieved by requiring nodes whose L-bit input value matches the agreed hashed value to deliver their L-bit input value to the other nodes jointly. By performing initial consensus only for the smaller hashed values, this algorithm is able to achieve communication complexity *linear* in n, i.e., $O(nL)$, for sufficiently large L and up to $f < n/2$ failures, with a non-zero probability of error.

Beerliova-Trubiniova and Hirt have presented an error-free linear communication complexity multi-party computation algorithm, which uses a linear complexity Byzantine *broadcast* algorithm as a sub-algorithm [31]. Their algorithm uses the idea of coding to reduce communication complexity, as does the Byzantine broadcast algorithm CBB we present in this chapter. Our broadcast algorithm CBB improves on their algorithm: while both algorithms have a similar structure, the communication complexity of the broadcast algorithm from [31] is 2 to 4 times as high as that of CBB, depending on the actual values of n and f. The main difference between the two algorithms is in the manner in which a code is used for error detection. In addition, the algorithm from [31] uses a *player elimination* framework, which is motivated by the *dispute control* framework proposed in [37]. In player elimination, when two nodes disagree with each other, one of the two nodes must be fault-free. Then both the nodes are removed from the system, and the underlying algorithm is performed on the smaller system, which must now tolerate one fewer faulty node, with two fewer nodes. This approach has also been adopted by asynchronous Byzantine agreement algorithms (e.g. [38]). While it may be possible to also use *player elimination* to achieve consensus, we believe that

the approach we adopted for the design of our Byzantine consensus algorithm CBC, in general, can more efficiently achieve stronger validity properties than approaches that may be designed using player elimination.

Practical BFT: Efforts have been devoted to make BFT practical. Castro and Liskov's practical Byzantine fault-tolerant (PBFT) state-machine replication algorithm [23] showed for the first time that BFT can be made practical. PBFT adopts the client-server model: the clients submit their requests to the servers, then the servers execute the requests and deliver the outcomes to the clients. One designated server is called the *"primary"* (or source in our terminology). The clients send their requests to the primary. Then the primary authenticates and orders the requests. The ordered requests are then broadcast to the other replicas (peers in our terminology) from the primary. In order to make sure that no two fault-free replicas accept different requests, hash values computed from the requests are exchanged among the replicas. Due to the use of hashing, the communication complexity is significantly reduced (to roughly $O(nL)$ for broadcasting L bits). Follow-ups of PBFT, such as Zyzzyva [24] and Aardvark [39], all take the similar hashing approach. However, due to the use of hashing, these algorithms are not error-free. The probability of error depends on the probability of collision of the hash function they use, and also on the adversary's ability to break it.

In designing the proposed algorithms, we drew inspiration from well-known ideas in prior work, as summarized next.

System-level diagnosis: Preparata, Metze and Chien [40] introduced the system *diagnosability* problem in their 1967 paper. Since then there has been a large body of work exploring different variations of the problem (e.g., [41]). The work on system-level diagnosis considers a (un)directed *diagnosis graph* (or a *test graph*), wherein each (un)directed edge represents a *test*: in essence, when node X tests node Y, it may declare Y as *faulty* or *fault-free*, with a faulty tester providing potentially erroneous test outcomes. The goal then is to use the results of the tests to either exactly identify the faulty nodes, or identify a small set of nodes that contains the faulty nodes. The past works differ in the nature of tests being performed, and the nature of the faults being diagnosed. In the system-level diagnosis jargon, our faults are *intermittent*

[42], and the tests are *comparison-based* [43]. We interpret the test outcome *fault-free* (*faulty*) as equivalent to the corresponding two nodes trusting (not trusting) each other, as discussed in detail later. In our work, we strengthen the comparison-based system-level diagnosis approach by incorporating an error detection code, which provides additional structure to our "comparison test" outcomes. This structure can be exploited for computational efficiency as well (see Appendix A.1).

Linear coding and block coding: A standard mechanism for improving efficiency of information transmission is to use *block codes*, meaning that a "block" (or multiple bits) of data is encoded together in a single codeword. Our specific approach for using linear error detection (block) codes for Byzantine consensus and broadcast is motivated by the rich literature on network coding, particularly, multicasting in the presence of a Byzantine attacker (e.g., [44, 45, 46]). Application of such an approach to Byzantine consensus or broadcast in an arbitrary point-to-point networks under per-link capacity constraints is non-trivial [10, 11]. However, under the *communication complexity* model, the problem is simpler, as the algorithms in this chapter demonstrate. Essentially, the simplification arises from the ability to treat each point-to-point link identically, resulting in a solution that has a certain symmetry. As we will see in the next chapter, such a symmetric solution is generally not optimal when the different links have different capacities.

Make the common case fast: In fault-tolerant systems, a common trick to improve average system performance (or reduce average overhead of fault-tolerance) is to make the "common case", namely, the failure-free execution, efficient, with the possibility of much higher overhead when a failure does occur. This approach works well when failure rates are low. There are many instances of the application of this idea, but some examples include error detection followed by retransmission for link reliability, and checkpointing and rollback or roll-forward recovery after failure detection [47].

2.4 Overview of the Proposed Algorithms

The proposed Byzantine consensus and broadcast algorithms are designed to perform efficiently for large-sized inputs, i.e., $L \gg 1$. Consequently, our discussion will assume that L is "sufficiently large" (how large is "sufficiently large" will become clearer later in this chapter). We now briefly describe the salient features of the consensus algorithm, with the detailed algorithm presented later in Sections 2.5 and 2.7.

Execution in Multiple Generations

To improve the communication complexity, consensus (or broadcast) for the L-bit value is performed "in parts". In particular, for a certain integer D, the L-bit value is divided into L/D parts, each consisting of D bits. For convenience of presentation, we will assume that L/D is an integer. A sub-algorithm is used to perform consensus (or broadcast) on each of these D-bit values, and we will refer to each execution of the sub-algorithm as a "generation".

Memory Across Generations

If during any one generation, misbehavior by some faulty node is detected, then additional (and expensive) diagnostic steps are performed to gain information on the potential identity of the misbehaving node(s). This information is captured by means of a *diagnosis graph*, as elaborated later. As the sub-algorithm is performed for each new generation, the *diagnosis graph* is updated to incorporate any new information that may be learned regarding the location of the faulty nodes. The execution of the sub-algorithm in each generation is adapted to the state of the diagnosis graph at the start of the generation.

Bounded Instances of Misbehavior

With Byzantine failures, it is not always possible to immediately determine the identity of a misbehaving node. However, due to the manner in which the diagnosis graph is maintained, and the manner in which the sub-algorithm

adapts to the diagnosis graph, the f (or fewer) faulty nodes can collectively misbehave in at most $f(f+1)$ generations, before all the faulty nodes are exactly identified. Once a faulty node is identified, it is effectively isolated from the network, and cannot tamper with future generations. Thus, $f(f+1)$ is also an upper bound on the number of generations in which the expensive diagnostic steps referred to above may need to be performed.

Low-Cost Failure-Free Execution

Due to the bounded number of generations in which the faulty nodes can misbehave, it turns out that the faulty nodes do not tamper with the execution in a majority of the generations. We use a low-cost mechanism to achieve consensus and broadcast in failure-free generations, which helps to achieve low communication complexity. In particular, we use an *error detection code*-based strategy in both algorithms to reduce the amount of information the nodes must exchange to be able to achieve consensus and broadcast in the absence of any misbehavior (the strategy, in fact, also allows detection of potential misbehavior).

Consistent Diagnosis Graph Maintenance

A copy of the diagnosis graph is maintained locally by each fault-free node. To ensure consistent maintenance of this graph, the *diagnostic information* (elaborated later) needs to be distributed consistently to all the nodes in the network. This operation is performed using an error-free 1-bit Byzantine broadcast algorithm that tolerates $f < n/3$ Byzantine failures with communication complexity of $O(n^2)$ bits [35, 34]. This 1-bit broadcast algorithm is referred to as `Broadcast_Binary` in our discussion. While `Broadcast_Binary` is expensive, its cumulative overhead is kept low by invoking it a relatively small number of times.

The structure above is inspired by prior work on fault-tolerant computing and communications theory, which is discussed in Section 2.3. It turns out that the "dispute control" approach used in the work on multi-party computation (MPC) has also used this structure [37], and its variation called

player elimination has been used to design an error-free linear complexity Byzantine *broadcast* algorithm [31].

We now elaborate on the error-detection code used in the proposed algorithms, and also describe the *diagnosis graph* in some more detail.

Error Detection Code

We will use Reed-Solomon codes in our algorithms (potentially, other codes may be used instead). Consider an (m, d) Reed-Solomon code in Galois Field $GF(2^c)$, where c is chosen large enough (specifically, $m \leq 2^c - 1$). This code encodes d data symbols from $GF(2^c)$ into a codeword consisting of m symbols from $GF(2^c)$. Each symbol from $GF(2^c)$ can be represented using c bits. Thus, a data vector of d symbols contains dc bits, and the corresponding codeword contains mc bits. Each symbol of the codeword is computed as a linear combination of the d data symbols, such that every subset of d coded symbols represents a set of linearly independent combinations of the d data symbols. This property implies that any subset of d symbols from the m symbols of a given codeword can be used to determine the corresponding data vector. Similarly, knowledge of any subset of d symbols from a codeword suffices to determine the remaining symbols of the codeword. So d is also called the *dimension* of the code. The (m, d) code has the Hamming distance of $m-d+1$, and can always detect up to $m-d$ errors. We will denote a code with dimension d as C_d, and the encoding/decoding operations as $Z = C_d(x)$ and $x = C_d^{-1}(Z)$ for a data vector x and the corresponding codeword Z.

In our algorithms, we also assume the availability of a null ($\perp$) symbol that is distinguished from all other symbols. For an m-element vector X, we denote $X[j]$ as the j-th element of the vector, $1 \leq j \leq m$. Given a subset $A \subseteq \{1, \ldots, m\}$, denote $V|A$ as the ordered list of elements of X at the locations corresponding to elements of A. For instance, if $m = 5$ and $A = \{2, 4\}$, then $X|A$ is equal to $(X[2], X[4])$. We will say that $X|A \in C_d$ if $X|A$ contains at least d non-null ($\neq\perp$) elements, and there exists a codeword $Z = C_d(x)$ for some x such that the non-null elements of $X|A$ are equal to the corresponding elements of $Z|A$. When such Z and x exist, by generalizing the decoding operation, we define $C_d^{-1}(X|A) = x$. If no such Z and x exist,

we will say that $X|A \notin C_d$.

For our consensus algorithm CBC, we use an $(n, v-f)$ distance-$(n-v+f+1)$ code C_{v-f}, for a suitable v such that $f+1 \leq v \leq n-f$, and ensure that there are at least $v-f$ non-null elements in the argument to C_{v-f}^{-1} (when the decoding function is used at a fault-free node).

For an improved version of CBC, we use an $(n, n-f)$ distance-$(f+1)$ code C_{n-f}. For the broadcast algorithm CBB, a $(2(n-1), n-f)$ distance-$(n+f-1)$ code is used. As will become clear from our later discussion, the key of our proofs rely only on the dimension, but not the length, of the code used. So to simplify the discussion, with a little abuse of notation, we also denote both code used in the improved version of the consensus algorithm CBC and the code used in the broadcast algorithm CBB as C_{n-f}, and ensure that there are at least $n-f$ non-null elements in the argument to C_{n-f}^{-1}. The code that we are referring to using C_{n-f} will be clear from the context.

Diagnosis Graph

The fault-free nodes' (potentially partial) knowledge of the identity of the faulty nodes is captured by a diagnosis graph. A diagnosis graph is an undirected graph with n vertices, with vertex i corresponding to node i. A pair of nodes are said to "trust" each other if the corresponding pair of vertices in the diagnosis graph is connected with an edge; otherwise they are said to "accuse" each other. In the dispute control jargon, two nodes that trust each other are "not in dispute", and two nodes that accuse each other are "in dispute".

Before the start of the very first generation, the diagnosis graph is initialized as a fully connected graph, which implies that all the n nodes initially trust each other. During the execution of the algorithm, whenever misbehavior by some faulty node is detected, the diagnosis graph will be updated, and one or more edges will be removed from the graph, using the diagnostic information communicated using the `Broadcast_Binary` algorithm. The use of `Broadcast_Binary` ensures that the fault-free nodes always have a consistent view of the diagnosis graph. The evolution of the diagnosis graph satisfies the following properties:

- If an edge is removed from the diagnosis graph, at least one of the

nodes corresponding to the two endpoints of the removed edge must be faulty.

- The fault-free nodes always trust each other.
- If more than f edges at a vertex in the diagnosis graph are removed, then the node corresponding to that vertex must be faulty.

The last two properties above follow from the first property, and the assumption that at most f node are faulty.

2.5 CBC – A Coding-Based Byzantine Consensus Algorithm

In this section, we describe CBC – a Byzantine consensus algorithm that has the properties listed in Section 2.1 – and present a proof of correctness. Also algorithm parameterization to satisfy more general *validity* requirements will be discussed in Section 2.5.5.

The L-bit input value x_i at each node is divided into L/D parts of size D bits each, as noted earlier. These parts are denoted as $x_i(1), \cdots, x_i(L/D)$. The algorithm for achieving L-bit consensus consists of L/D sequential executions of Algorithm 2.5.1 presented in this section. Algorithm 2.5.1 is executed once for each generation. For the g-th generation ($1 \leq g \leq L/D$), each node i uses $x_i(g)$ as its input in Algorithm 2.5.1. Each generation of the algorithm results in node i deciding on g-th part (namely, $y_i(g)$) of its final decision value y_i.

The value $x_i(g)$ is represented by a vector of $n-2f$ symbols, each symbol represented with $D/(n-2f)$ bits. For convenience of presentation, assume that $D/(n-2f)$ is an integer. We will refer to these $n-2f$ symbols as the *data symbols*.

An $(n, n-2f)$ distance-$(2f+1)$ Reed-Solomon code, denoted as C_{n-2f}, is used to encode the $n-2f$ data symbols into n coded symbols. We assume that $D/(n-2f)$ is large enough to allow the above Reed-Solomon code to exist, specifically, $n \leq 2^{D/(n-2f)} - 1$, which implies that $D = \Omega(n \log n)$. This condition is met only if L is large enough (since $L > D$). As we will see later, C_{n-2f} is used only for error detection.

Let the set of all the fault-free nodes be denoted as P_{good}. Algorithm 2.5.1 for each generation g consists of three stages. We summarize the function of these three stages first, followed by a more detailed discussion:

1. Matching stage: Each node i encodes its D-bit input $x_i(g)$ for generation g into n coded symbols, as noted above. Each node i sends one of these n coded symbols to the other nodes *that it trusts*. Node i trusts node j if and only if the corresponding vertices in the diagnosis graph are connected by an edge. Using the symbols thus received from each other, the nodes attempt to identify a "matching set" of nodes, denoted as P_{match}, of size $n-f$ such that the fault-free nodes in P_{match} are guaranteed to have an identical input value for the current generation. If such a P_{match} is not found, it can be determined with certainty that all the fault-free nodes do not have the same input value – in this case, the fault-free nodes decide on a default output value and terminate the algorithm.

2. Checking stage: If a set of nodes P_{match} is identified in the above matching stage, each node $j \notin P_{match}$ checks whether the symbols received from nodes in P_{match} correspond to a valid codeword. If such a codeword exists, then the symbols received from P_{match} are said to be "consistent". If any node finds that these symbols are not consistent, then misbehavior by some faulty node is detected. Else all the nodes are able to correctly compute the value to be agreed upon in the current generation.

3. Diagnosis stage: Whenever misbehavior is detected, the diagnosis stage is performed, to learn (possibly partial) information regarding the identity of the faulty node(s). For fault diagnosis, the nodes in P_{match} are required to *broadcast* the coded symbols they sent in the matching stage, using the `Broadcast_Binary` algorithm. Using the information received during these broadcasts, the fault-free nodes are able to learn new information regarding the potential identity of the faulty node(s). The *diagnosis graph* (called Diag_Graph in Algorithm 2.5.1) is updated to incorporate this new information.

In the rest of this section, we discuss each of the three stages in more detail. Note that whenever algorithm `Broadcast_Binary` is used, all the

fault-free nodes will receive the broadcast information identically. One instance of `Broadcast_Binary` is needed for each bit of information broadcast using `Broadcast_Binary`.

Algorithm 2.5.1 CBC Algorithm (generation g)

1.**Matching Stage:**

Each node i performs the matching stage as follows:

(a)Compute $(S_i[1], \ldots, S_i[n]) = C_{n-2f}(x_i(g))$, and *send* $S_i[i]$ to every trusted node j

(b)$R_i[j] \leftarrow \begin{cases} \text{symbol that node } i \text{ receives from node } j, \\ \qquad\qquad\qquad\qquad \text{if node } i \text{ trusts node } j; \\ \perp, \text{ otherwise.} \end{cases}$

If a trusted node j does not send anything, then $R_i[j] \leftarrow 0$.

(c)If $S_i[j] = R_i[j]$ then $M_i[j] \leftarrow$ **TRUE**; else $M_i[j] \leftarrow$ **FALSE**

(d)Node i broadcasts the vector M_i using `Broadcast_Binary`

Using the received M vectors:

(e)Find a set of nodes P_{match} of size $n - f$ such that

$$M_j[k] = M_k[j] = \textbf{TRUE}$$

for every pair of nodes $j, k \in P_{match}$. If multiple possibility exist for P_{match}, then any one of the possible sets is chosen arbitrarily as P_{match} (all fault-free nodes choose a deterministic algorithm to select identical P_{match}).

(f)If P_{match} does not exist, then decide on a default value and terminate;
else enter the Checking Stage

2.**Checking Stage:**

Each node $j \notin P_{match}$ performs steps 2(a) and 2(b):

(a)If $R_j|P_{match} \in C_{n-2f}$ then $Detected_j \leftarrow$ **FALSE**;
else $Detected_j \leftarrow$ **TRUE**.

(b)Broadcast $Detected_j$ using `Broadcast_Binary`

Each node i performs step 2(c):

(c)Receive $Detected_j$ from each node $j \notin P_{match}$ (broadcast in step 2(b)).
If $Detected_j =$ **FALSE** for all nodes $j \notin P_{match}$, then decide on

$$y_i(g) = C^{-1}_{n-2f}(R_i|P_{match});$$

else enter Diagnosis Stage

3.**Diagnosis Stage:**

Each node $j \in P_{match}$ performs step 3(a):

(a)Broadcast $S_j[j]$ using `Broadcast_Binary`

Each node i performs the following steps:

(b)$R^{\#}[j] \leftarrow$ symbol received from node $j \in P_{match}$ as a result of broadcast in step 3(a)

(c)For all nodes $j \in P_{match}$,
if node i trusts node j and $R_i[j] = R^{\#}[j]$ then

$$Trust_i[j] \leftarrow \textbf{TRUE};$$

else $Trust_i[j] \leftarrow$ **FALSE**

(d)Broadcast $Trust_i|P_{match}$ using `Broadcast_Binary`

(e)For each edge (j, k) in Diag_Graph, such that node $j \in P_{match}$, remove edge (j, k)
if $Trust_j[k] =$ **FALSE** or $Trust_k[j] =$ **FALSE**

(f)If $R^{\#}|P_{match} \in C_{n-2f}$ then
if for any node $j \notin P_{match}$, $Detected_j =$ **TRUE**, but no edge at vertex j was removed in step 3(e), then
remove all edges at vertex j in Diag_Graph

(g)If at least $f + 1$ edges at any vertex j have been removed so far, then node j must be faulty, and all edges at j are removed.

(h)Find a set of nodes $P_{decide} \subset P_{match}$ of size $n - 2f$ in the updated Diag_Graph,
such that every pair of nodes $j, k \in P_{decide}$ still trust each other

(i)Decide on $y_i(g) = C_{n-2f}^{-1}(R^{\#}|P_{decide})$

NOTE: Instead of performing steps 3(h) and 3(i), Algorithm 2.5.1 may be repeated for generation g again after the diagnosis graph has been updated in step 3(g). This alternative does not affect algorithm correctness.

2.5.1 Matching Stage

The line numbers referred to below correspond to the line numbers for the pseudo-code in Algorithm 2.5.1.

`Line 1(a):` In generation g, each node i first encodes $x_i(g)$, represented by $n-2f$ symbols, into a codeword S_i from the code C_{n-2f}. The j-th symbol in the codeword is denoted as $S_i[j]$. Then node i sends $S_i[i]$, the i-th symbol of its codeword, to all the other nodes *that it trusts*. Recall that node i trusts node j if and only if there is an edge between the corresponding vertices in the diagnosis graph (referred as Diag_Graph in the pseudo-code).

`Line 1(b):` Let us denote by $R_i[j]$ the symbol that node i receives from a trusted node j. If a node i does not trust some node j, then node i sets $R_i[j]$ equal to null ($\perp$). Messages received from untrusted nodes are ignored.

`Line 1(c)`: Flag $M_i[j]$ is used to record whether node i finds node j's symbol consistent with its own local value. Specifically, the pseudo-code in Line 1(c) is equivalent to the following:

- When node i trusts node j:
  If $R_i[j] = S_i[j]$, then $M_i[j] =$ **TRUE**;
  else $M_i[j] =$ **FALSE**.
- When node i does not trust node j: $M_i[j] =$ **FALSE**.

`Line 1(d):` As we will see later, if a fault-free node i does not trust another node j, then node j must be faulty. Thus entry $M_i[j]$ in vector M_i is **FALSE**

if (i) node i believes that node j is faulty, or (ii) the input value at node j appears to differ from the input value at node i. Thus, entry $M_i[j]$ being **TRUE** implies that, as of this time, node i believes that node j is fault-free, and that the value at node j is possibly identical to the value at node i. Node i uses `Broadcast_Binary` to broadcast M_i to all the nodes. One instance of `Broadcast_Binary` is needed for each bit of M_i.

`Lines 1(e) and 1(f):` Due to the use of `Broadcast_Binary`, all fault-free nodes receive identical vector M_j from each node j. Using these M vectors, each node i attempts to find a set P_{match} containing $n - f$ nodes such that, for every pair of nodes $j, k \in P_{match}$, $M_j[k] = M_k[j] =$ **TRUE**. If multiple such sets P_{match} exist, the tie is broken by some predetermined function. One example of the predetermined function is: map the set P_{match} into an n-bit binary value such that the i-th bit is 1 if node $i \in P_{match}$, and 0 otherwise; then the one set with smallest binary value is picked as P_{match} in the algorithm. Since the M vectors are received identically by all the fault-free nodes (using `Broadcast_Binary`), they can compute an identical P_{match}. However, if such a set P_{match} does not exist, then the fault-free nodes conclude that all the fault-free nodes do not have identical input – in this case, they decide on a default value, and terminate the algorithm.

It is worth noting that finding P_{match} is, in fact, equivalent to identifying a clique of size $n - f$ in an undirected graph of size n, whose edges are defined by the M vectors. Specifically, an edge exists between j and k in this graph, if $M_j[k] = M_k[j] =$ **TRUE**. Finding a clique of a certain size in general graphs is NP-complete. It turns out that, with a slight modification, the algorithm can perform correctly even if P_{match} is not a clique in the graph induced by M vectors. Instead it suffices if P_{match} includes at least $n - 2f$ fault-free nodes with identical input for the current generation. In other words, the subgraph induced by M and P_{match} will contain a clique of size $n-2f$ corresponding to fault-free nodes. Such a P_{match} can be found with polynomial computational complexity (see Appendix A.1). However, for simplicity, in our proofs, we will assume that P_{match} indeed corresponds to a clique of size $n - f$.

In the following discussion, we will show the correctness of the *Matching Stage*. In the proofs of Lemmas 2.1, 2.2, and 2.3, we assume that the fault-free nodes (that is, the nodes in set P_{good}) always trust each other – this

assumption will be shown to be correct later in Lemma 2.4.

Lemma 2.1 *If for each fault-free node $i \in P_{good}$, $x_i(g) = x(g)$, for some value $x(g)$, then a set P_{match} necessarily exists (assuming that the fault-free nodes trust each other).*

Proof: Since all the fault-free nodes have identical input $x(g)$ in generation g, $S_i = C_{n-2f}(x(g))$ for all nodes $i \in P_{good}$. Since these nodes are fault-free, and trust each other, they send to each other correct messages in the matching stage. Thus, $R_i[j] = S_j[j] = S_i[j]$ for all nodes $i, j \in P_{good}$. This fact implies that $M_i[j] =$ **TRUE** for all nodes $i, j \in P_{good}$. Since there are at least $n - f$ fault-free nodes, it follows that a set P_{match} of size $n - f$ must exist.

□

Observe that, although the above proof shows that there exists a set P_{match} containing only fault-free nodes, there may also be other such sets that contain some faulty nodes as well. That is, all the nodes in P_{match} cannot be assumed to be fault-free. The converse of Lemma 2.1 implies that, if a set P_{match} does not exist, it is certain that all the fault-free nodes do not have the same input values. In this case, they can correctly agree on a default value and terminate the algorithm. Thus Line 1(f) is correct.

In the case when a set P_{match} is found, the following lemma is useful.

Lemma 2.2 *All nodes in $P_{match} \cap P_{good}$ have identical input in generation g.*

Proof: $|P_{match} \cap P_{good}| \geq n - 2f$ because $|P_{match}| = n - f$ and there are at most f faulty nodes. Consider any two nodes $i, j \in P_{match} \cap P_{good}$. Since $M_i[j] = M_j[i] =$ **TRUE**, it follows that $S_i[i] = S_j[i]$ and $S_j[j] = S_i[j]$. Since there are $n-2f$ fault-free nodes in $P_{match} \cap P_{good}$, this implies that the codewords computed by these fault-free nodes (in Line 1(a)) contain at least $n-2f$ identical symbols. Since the code C_{n-2f} has dimension $(n-2f)$, this implies that the fault-free nodes in $P_{match} \cap P_{good}$ must have identical input in generation g.

□

2.5.2 Checking Stage

When P_{match} is found during the matching stage, the checking stage is entered.

`Lines 2(a), 2(b):` Each fault-free node $j \notin P_{match}$ checks whether the symbols received from the trusted nodes in P_{match} are consistent with a valid codeword: that is, check whether $R_j|P_{match} \in C_{n-2f}$. The result of this test is broadcast as a 1-bit notification $Detected_j$, using `Broadcast_Binary`. If $R_j|P_{match} \notin C_{n-2f}$, then node j is said to have detected an *inconsistency*.

`Line 2(c):` If no node announces in Line 2(b) that it has detected an inconsistency, each fault-free node i chooses $C^{-1}_{n-2f}(R_i|P_{match})$ as its output for generation g.

The following lemma argues correctness of Line 2(c).

Lemma 2.3 *If no node claims it has detected an inconsistency in Lines 2(a)-2(b), all fault-free nodes $i \in P_{good}$ decide on the identical output value $y(g)$ such that $y(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$.*

Proof: We assume that the fault-free nodes trust each other. Observe that size of set $P_{match} \cap P_{good}$ is at least $n-2f$, and hence the decoding operations $C^{-1}_{n-2f}(R_i|P_{match})$ and $C^{-1}_{n-2f}(R_i|P_{match} \cap P_{good})$ are both defined at fault-free nodes.

Since fault-free nodes send correct messages, and trust each other, for all nodes $i \in P_{good}$, $R_i|P_{match} \cap P_{good}$ are identical. Since no inconsistency has been detected by any node, every node $i \in P_{good}$ decides on $C^{-1}_{n-2f}(R_i|P_{match})$ as its output. Also, $C^{-1}_{n-2f}(R_i|P_{match}) = C^{-1}_{n-2f}(R_i|P_{match} \cap P_{good})$, since C_{n-2f} has dimension $(n-2f)$. It then follows that all the fault-free nodes decide on the identical value $y(g) = C^{-1}_{n-2f}(R_i|P_{match} \cap P_{good})$ in Line 2(c). Since $R_j|P_{match} \cap P_{good} = S_j|P_{match} \cap P_{good}$ for all nodes $j \in P_{match} \cap P_{good}$, $y(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$.

□

2.5.3 Diagnosis Stage

When any node that is not in P_{match} announces that it has detected an inconsistency, the diagnosis stage is entered. The algorithm allows for the

possibility that a faulty node may erroneously announce that it has detected an inconsistency. The purpose of the diagnosis stage is to learn new information regarding the potential identity of a faulty node. The new information is used to remove one or more edges from the diagnosis graph Diag_Graph – as we will soon show, when an edge (j, k) is removed from the diagnosis graph, at least one of nodes j and k must be faulty. We now describe the steps in the Diagnosis Stage.

`Lines 3(a), 3(b):` Every fault-free node $j \in P_{match}$ uses `Broadcast_Binary` to broadcast $S_j[j]$ to all nodes. Let us denote by $R^{\#}[j]$ the result of the broadcast from node j. Due to the use of `Broadcast_Binary`, all fault-free nodes receive identical $R^{\#}[j]$ for each node $j \in P_{match}$. This information will be used for diagnostic purposes.

`Lines 3(c), 3(d):` Every fault-free node i uses flag $Trust_i[j]$ to record whether it "*believes*", as of this time, that each node $j \in P_{match}$ is fault-free or not. Then node i broadcasts $Trust_i|P_{match}$ to all nodes using `Broadcast_Binary`. Specifically,

- If node i trusts node j **and** $R_i[j] = R^{\#}[j]$,
  then set $Trust_i[j]$ =**TRUE**;
- If node i does not trust node j **or** $R_i[j] \neq R^{\#}[j]$,
  then set $Trust_i[j]$ =**FALSE**.

`Line 3(e):` Using the $Trust$ vectors received above, each fault-free node i then removes any edge (j, k) from the diagnosis graph such that $Trust_j[k] =$ **FALSE** or $Trust_k[j] =$ **FALSE**. Due to the use of `Broadcast_Binary` for distributing $Trust$ vectors, all fault-free nodes will maintain an identical view of the updated Diag_Graph. Note that edges may only be removed from Diag_Graph.[1]

`Line 3(f):` As we will soon show, in the case $R^{\#}|P_{match} \in C_{n-2f}$, a node $j \notin P_{match}$ that announces that it has detected an inconsistency, i.e., $Detected_j =$

[1] If the system allows "repair" of faulty nodes, then edges will need to be added back to Diag_Graph.

TRUE, must be faulty if no edge attached to vertex j was removed in Line 3(e). Such a node j is "*isolated*", by having all edges attached to vertex j removed from Diag_Graph, and the fault-free nodes will not communicate with it anymore in subsequent generations.

`Line 3(g):` As we will soon show, a node j must be faulty if at least $f+1$ edges at vertex j have been removed. The identified faulty node j is then isolated.

`Lines 3(h) and 3(i):` Since Diag_Graph is updated only with information broadcast with `Broadcast_Binary` (*Detected*, $R^{\#}$ and $Trust$), all fault-free nodes maintain an identical view of the updated Diag_Graph. Then they can compute an identical set $P_{decide} \subset P_{match}$ containing exactly $n-2f$ nodes such that every pair of nodes $j, k \in P_{decide}$ trust each other. Finally, every fault-free node chooses $C_{n-2f}^{-1}(R^{\#}|P_{decide})$ as its decision value for generation g.

Lemma 2.4 *Every time the diagnosis stage is performed, at least one edge attached to a vertex corresponding to a faulty node will be removed from Diag_Graph, and only such edges will be removed.*

Proof: We prove this lemma by induction. For the convenience of discussion, let us say that an edge (j, k) is "*bad*" if at least one of nodes j and k is faulty.

Consider a generation g starting with any instance of the Diag_Graph in which only bad edges have been removed. Suppose that a node $i \notin P_{match}$ "claims" that it detects a failure by broadcasting $Detect_i =$**TRUE** in Line 2(b). This implies that, if node i is fault-free, $R_i|P_{match}$ cannot be a valid codeword, i.e., $R_i|P_{match} \notin C_{n-2f}$.

The symbols broadcast by nodes of P_{match} in Line 3(a), i.e., $R^{\#}|P_{match}$, can either be a valid codeword of C_{n-2f} or not. We consider the two possibilities separately:

- $R^{\#}|P_{match} \in C_{n-2f}$: In the case, if node i is actually fault-free, $R^{\#}[k] \neq R_i[k]$ must be true for some faulty node $k \in P_{match}$, which is trusted by node i. Thus, $Trust_i[k] =$ **FALSE** and the bad edge (i, k) will be removed in Line 3(e). The converse of this argument implies that

if $Detected_i$ =**TRUE** but no edge attached to vertex i is removed in Line 3(e), then node i must be faulty. As a result, all bad edges at vertex i are removed in Line 3(f).

- $R^{\#}|P_{match} \notin C_{n-2f}$: There are always at least $n-2f$ fault-free nodes in $P_{match} \cap P_{good}$, and $R_j|P_{match} \in C_{n-2f}$ is **TRUE** for every node $j \in P_{match} \cap P_{good}$. Thus, if $R^{\#}|P_{match} \notin C_{n-2f}$, then $R^{\#}|P_{match} \neq R_j|P_{match}$ must be true for every node $j \in P_{match} \cap P_{good}$. Similar to the previous case, $R^{\#}[k] \neq R_j[k]$ must be true for some faulty node $k \in P_{match}$ which is trusted by every node $j \in P_{match} \cap P_{good}$. As a result, $Trust_j[k] =$ **FALSE** and the bad edge (j,k) will be removed in Line 3(e), for all nodes $j \in P_{match} \cap P_{good}$.

At this point, we can conclude that by the end of Line 3(f), at least one new bad edge has been removed. Moreover, since $R_i[k] = R^{\#}[k]$ for every pair of fault-free nodes $i, k \in P_{good}$, $Trust_i[k]$ remains **TRUE**, which implies that the vertices corresponding to the fault-free nodes will remain fully connected, and each will always have at least $n-f-1$ edges. It follows that a node j must be faulty if at least $f+1$ edges at vertex j have been removed. So all edges at j are bad and will be removed in Line 3(g).

Now we have proved that for every generation that begins with a *Diag_Graph* in which only bad edges have been removed, at least one new bad edge, and only bad edges, will be removed in the updated *Diag_Graph* by the end of the diagnosis stage. Together with the fact that *Diag_Graph* is initialized as a complete graph, we finish the proof.

□

The above proof of Lemma 2.4 shows that all fault-free nodes will trust each other throughout the execution of the algorithm, which justifies the assumption made in the proofs of the previous lemmas. The following lemma shows the correctness of Lines 3(h) and 3(i).

Lemma 2.5 *By the end of diagnosis stage, all fault-free nodes $i \in P_{good}$ decide on the same output value $y(g)$, such that $y(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$.*

Proof: First of all, the set P_{decide} necessarily exists since there are at least $n-2f \geq f+1$ fault-free nodes in $P_{match} \cap P_{good}$ that always trust each other.

Secondly, since the size of P_{decide} is $n - 2f \geq f + 1$, it must contain at least one fault-free node $k \in P_{decide} \cap P_{good}$. Since node k still trusts all nodes of P_{decide} in the updated $Diag\_Graph$, $R^{\#}|P_{decide} = R_k|P_{decide} = S_k|P_{decide}$. The second equality is due to the fact that node $k \in P_{match}$. Finally, since the size of set P_{decide} is $n - 2f$, the decoding operation of $C^{-1}_{n-2f}(R^{\#}|P_{decide})$ is defined, and it equals to $x_k(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$, as per Lemma 2.2.

□

We can now conclude the correctness of the Algorithm 2.5.1.

Theorem 2.1 *Given n nodes with at most $f < n/3$ are faulty, each given an input value of L bits, Algorithm 2.5.1 achieves consensus correctly in L/D generations, with the diagnosis stage performed for at most $f(f + 1)$ times.*

Proof: Lemmas 2.1 to 2.5 imply that consensus is achieved correctly for each generation g of D bits. So the termination and consistency properties are satisfied for the L-bit outputs after L/D generations. Moreover, in the case all fault-free nodes are given an identical L-bit input x, the D bits output $y(g)$ in each generation g equals to $x(g)$ as per Lemmas 2.1, 2.3 and 2.5. So the L-bit output $y = x$ and the validity property is also satisfied.

According to Lemma 2.4 and the fact that a faulty node P_j will be removed once more than f edges at vertex j have been removed, it takes at most $f(f + 1)$ instance of the diagnosis stage before all faulty nodes are identified. After that, the fault-free nodes will not communicate with the faulty nodes. Thus, the diagnosis stage will not be performed any more. So it will be performed for at most $f(f + 1)$ times in all cases.

□

2.5.4 Communication Complexity of CBC

Let us denote by B the complexity of broadcasting 1 bit with one instance of `Broadcast_Binary`. In every generation, the complexity of each stage is as follows:

- Matching stage: every node i sends at most $n - 1$ symbols, each of $D/(n - 2f)$ bits, to the nodes that it trusts, and broadcasts $n - 1$ bits

for M_i. So at most $\frac{n(n-1)}{n-2f}D + n(n-1)B$ bits in total are transmitted by all n nodes.

- Checking stage: every node $j \notin P_{match}$ broadcasts one bit $Detected_j$ with `Broadcast_Binary`, and there are f such nodes. So tB bits are transmitted.

- Diagnosis stage: every node $j \in P_{match}$ broadcasts one symbol $S_j[j]$ of $D/(n-2f)$ bits with `Broadcast_Binary`, and every node i broadcasts $n-f$ bits of $Trust_i|P_{match}$ with `Broadcast_Binary`. So the complexity is $\frac{n-f}{n-2f}DB + n(n-f)B$ bits.

According to Theorem 2.1, there are L/D generations in total. In the worst case, P_{match} can be found in every generation, so the matching and checking stages will be performed for L/D times. In addition, the diagnosis stage will be performed for at most $f(f+1)$ time. Hence the communication complexity of the proposed consensus algorithm, denoted as $C_{CBC}(L)$, is then computed as

$$\begin{aligned} C_{CBC}(L) &= \left(\frac{n(n-1)}{n-2f}D + n(n-1)B + fB\right)\frac{L}{D} \\ &\quad + f(f+1)\left(\frac{n-f}{n-2f}D + n(n-f)\right)B. \end{aligned} \tag{2.1}$$

For a large enough value of L, with a suitable choice of $D = \sqrt{\frac{(n^2-n+f)(n-2f)L}{f(f+1)(n-f)}}$, we have

$$\begin{aligned} C_{CBC}(L) &= \frac{n(n-1)}{n-2f}L + f(f+1)n(n-f)B \\ &\quad + 2BL^{0.5}\sqrt{\frac{(n^2-n+f)f(f+1)(n-f)}{n-2f}}. \end{aligned} \tag{2.2}$$

Error-free algorithms that broadcast 1 bit with communication complexity $\Theta(n^2)$ bits are known [34, 35]. So we assume $B = \Theta(n^2)$. Then the complexity of our algorithm for $t < n/3$ becomes

$$\begin{aligned} C_{CBC}(L) &= \frac{n(n-1)}{n-2f}L + O(n^4L^{0.5} + n^6) \\ &= O(nL + n^4L^{0.5} + n^6). \end{aligned} \tag{2.3}$$

For $L = \Omega(n^6)$, the communication complexity becomes $O(nL)$. (This requirement can be improved to $L = \Omega(n^5)$ if we use a technique from [37] in the Diagnosis stage.[2])

2.5.5 Other Validity Conditions

Algorithm 2.5.1 satisfies the validity conditions stated in Section 2.1. As noted earlier, other validity conditions may also be desirable in practice. Algorithm 2.5.1 is flexible in the sense that, with proper parameterization, it can achieve other (reasonable) validity properties. In particular, v-validity property defined below can be achieved for $f + 1 \leq v \leq n - f$:

- v-Validity: If at least v fault-free nodes hold an identical input x, then the output y agreed by the fault-free nodes equals input x_j for some fault-free node j. Furthermore, if $v \geq \lceil \frac{n+1}{2} \rceil$, then $y = x$.

In order to achieve v-validity, we need to change the error detection code and the size of P_{match} in Algorithm 2.5.1 as follows (v is said to be the parameter of the algorithm):

- Throughout the algorithm, we replace the $(n, n-2f)$ distance-$(2f+1)$ code C_{n-2f} with a $(n, v-f)$ distance-$(n-v+f+1)$ code, denoted as C_{v-f}. Since $v \geq f+1$, $v - f \geq 1$. Thus C_{v-f} always exists for large enough L.

- `Line 1(e):` Choose a set of v nodes P_{match} such that $M_j[k] = M_k[j] =$ **TRUE** for every pair of node $j, k \in P_{match}$. [3]

- `Lines 3(h) and 3(i):` For the more general validity conditions, instead of performing steps 3(h) and 3(i), as stated in the NOTE at the end of Algorithm 2.5.1, the algorithm can be repeated for generation g with the updated diagnosis graph. For $v > 2f$, we also have the alternative of retaining steps 3(h) and 3(i), but the size of the chosen P_{decide} must be $v - f$.

Similar to Lemmas 2.1 through 2.5, we can prove that

[2] We would like to thank Martin Hirt for suggesting this improvement.

[3] The validity condition achieved can be made stronger, particularly for $v \leq n/2$, by choosing the largest possible set P_{match} with size at least v.

1. If at least v fault-free nodes $i \in P_{good}$ hold the same input $x_i(g) = x(g)$ for some $x(g)$, then a set P_{match} of size v necessarily exists.

2. There are at least $v - f \geq 1$ fault-free nodes in P_{match} and all the fault-free nodes in P_{match} have the same input for generation g.

3. If no node detects inconsistency in Line 2(a), all fault-free nodes $i \in P_{good}$ decide on the identical output value $y(g)$ such that $y(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$. Furthermore, when $v \geq \lceil \frac{n+1}{2} \rceil$, and at least v fault-free nodes have identical input x, at least one of these fault-free nodes is bound to be in P_{match}, and therefore, $y(g) = x$.

4. In case (for $v > 2f$) steps 3(h) and 3(i) are used, by the end of diagnosis stage, all fault-free nodes $i \in P_{good}$ decide on the same output value $y(g)$, such that $y(g) = x_j(g)$ for all nodes $j \in P_{match} \cap P_{good}$. Otherwise, if steps 3(h) and 3(i) are not used, the algorithm is repeated for generation g with the updated diagnosis graph.

Then Algorithm 2.5.1 achieves v-validity for $f+1 \leq v \leq n-f$ with the aforementioned parameterization. The communication complexity for achieving v-validity is

$$\begin{aligned} C^v_{CBC}(L) &= \frac{n(n-1)}{v-f}L + O(n^4 L^{0.5} + n^6) \\ &= O(nL + n^4 L^{0.5} + n^6), \text{ if } v - f = \Omega(n). \end{aligned}$$

When $v < \lceil \frac{n+1}{2} \rceil$, there may be more than one choice for P_{match} in Line 1(e). For example, there can be two disjoint cliques, one containing v fault-free nodes with the same input x, and the other containing $v - f$ fault-free nodes with input z ($x \neq z$) and f faulty nodes that pretend to have input z. It is possible that the second clique is picked to be P_{match} and the consensus value ends up being z. So in this case, even though at least v fault-free nodes hold the same input, we can only guarantee agreement on the input of *some* fault-free nodes, but not necessarily equal to the v identical inputs. However, when $v \geq \lceil \frac{n+1}{2} \rceil$, it is guaranteed that, if at least v inputs at the fault-free nodes equals to x, then the agreed output equals x.

2.5.6 Multiple Consensus

In the above discussion, we considered consensus on a single long L-bit value. An alternate view of the problem is likely to be more relevant in practice. In particular, let us consider the problem of performing g instances of consensus, with each node receiving a D-bit input for each instance (in particular, the input for node i is $x_i(g)$ for instance g). Then the consensus properties need to be satisfied for each instance *separately*. We assume that the identity of faulty nodes remains fixed across the different instances. Then it should not be difficult to see that Algorithm 2.5.1 solves the multiple instances of the consensus problem, with the algorithm for g-th generation essentially performing the g-th instance of the consensus problem for D-bit values.

Let us denote the *average* per-instance complexity for performing g instances of consensus on D-bit values as $\overline{C}_{CBC}(g, D)$. Similar to the analysis in Section 2.5.4, for $g \geq f(f+1)$, we have

$$\begin{aligned}
\overline{C}_{CBC}(g,D) &= \frac{n(n-1)}{n-2f}D + n(n-1)B + tB \\
&\quad + \frac{f(f+1)}{g}\left(\frac{n-f}{n-2f}D + n(n-f)\right)B \qquad (2.4)\\
&= O\left(\left(n + \frac{n^4}{g}\right)D + n^4 + \frac{n^6}{g}\right). \qquad (2.5)
\end{aligned}$$

From Equation 2.5, we can conclude that we only need $D = \Omega(n^3)$ and $g = \Omega(n^3)$ instances of D-bit consensus for the per consensus complexity to reduce to $O(nD)$. In other words, when the goal is to sequentially perform a large number ($\Omega(n^3)$) of instances of consensus, the input size for each instance only needs to be $\Omega(n^3)$ in order to achieve complexity linear in n, rather than $\Omega(n^6)$ as discussed in Section 2.5.4.

2.6 Improving the CBC Algorithm

2.6.1 Improving Communication Complexity of CBC

The CBC algorithm has complexity $\frac{n(n-1)}{n-2f}L$ (ignoring the terms sub-linear in L). In this section, we present Improved-CBC, an improved version of CBC that achieves communication complexity $\frac{n(n-1)}{n-f}L$, which can be twice

as efficient as CBC when f gets close to $n/3$. Pseudo-code of the Improved-CBC algorithm is presented in Algorithm 2.6.2.

The main difference between CBC and Improved-CBC is the way in which P_{match} is found. In CBC, a set P_{match} is found in each generation independently, and P_{match} from different generations can contain different nodes. On the other hand, P_{match} is maintained across generations in Improved-CBC such that P_{match} in generation $g+1$ is always a subset of P_{match} in generation g. In generation 1, P_{match} is initialized to the set of all n nodes in Improved-CBC.

Algorithm 2.6.2 Improved-CBC Algorithm (generation g)

In the following steps, for every node i: $R_i[k] \leftarrow S_j[k]$ whenever node i receives $S_j[k]$ from its trusted node j. If a trusted node j does not send anything when it should send $R_j[k]$, then $R_i[k] \leftarrow 0$.

1. **Matching Stage:**

   Each node $P_i \in P_{match}$ performs steps 1(a) and 1(b) as follows:

   (a) Compute $(S_i[1], \ldots, S_i[n]) = C_{n-f}(x_i(g))$, and *send* $S_i[i]$ to every trusted node j (including those not in P_{match}, and node i itself).

   (b) For every node $j \notin P_{match}$ that is trusted by node i:
   If $i = \min\{l | l \in P_{match}$ and node j trusts node $l\}$, then node i sends $S_i[k]$ to node j for each $k \in P_{match}$ such that node j does not trust node k.

   Each node $j \notin P_{match}$ performs step 1(c) as follows:

   (c) Using the first $n-f$ symbols it has received in steps 1(a) and 1(b), node j computes $S_j[j]$ according to C_{n-f}, then sends $S_j[j]$ to all trusted nodes (including node j itself).

2. **Checking Stage:**

   Each node i (in P_{match} or not) performs Checking Stage as follows:

   (a) If $R_i \in C_{n-f}$ then $Detected_i \leftarrow$ **FALSE**; else $Detected_i \leftarrow$ **TRUE**.

   (b) If node $i \in P_{match}$ and $R_i \neq S_i$ then $Detected_i \leftarrow$ **TRUE**.

   (c) Broadcast $Detected_i$ using `Broadcast_Binary`.

(d)Receive $Detected_j$ from each node j (broadcast in step 2(c)).
If $Detected_j =$ **FALSE** for all node j, decide on $y_i(g) = C_{n-f}^{-1}(R_i)$; else enter Diagnosis Stage.

3.**Diagnosis Stage:**

Each node i (in P_{match} or not) performs Diagnosis Stage as follows:

(a)Broadcast S_i and R_i using `Broadcast_Binary`.

(b)$S_j^{\#} \leftarrow S_j$ and $R_j^{\#} \leftarrow R_j$ received from node j as a result of broadcast in step 3(a).

Using the broadcast information, all nodes perform the following steps identically:

(c)For each edge (i, j) in Diag_Graph: Remove edge (i, j) if $\exists k$, such that node j receives $S_i[k]$ from node i in Matching stage and $R_j^{\#}[k] \neq S_i^{\#}[k]$

(d)For each node $i \in P_{match}$: If $S_i^{\#} \notin C_{n-f}$, then node i must be faulty. So remove vertex i and the adjacent edges from Diag_Graph.

(e)For each node $j \notin P_{match}$: If $S_j^{\#}[j]$ is not consistent with (according to C_{n-f}) the subset of $n-f$ symbols of $R_j^{\#}$, from which $S_j^{\#}[j]$ is computed, node j must be faulty. So remove vertex j and the adjacent edges from Diag_Graph.

(f)If at least $f+1$ edges at any vertex i have been removed, then node i must be faulty. So remove vertex i and the adjacent edges.

(g)Find the maximum set of nodes $P_{new} \subseteq P_{match}$ such that $S_i^{\#} = S_j^{\#}$ for every pair of nodes $i, j \in P_{new}$. In case of a tie, pick any one.

(h)If $|P_{new}| < n - f$, terminate the algorithm and decide on the default output.
Else, decide on $y_i(g) = C_{n-f}^{-1}(S_j^{\#})$ for any node $j \in P_{new}$, and update $P_{match} = P_{new}$.

The proof of correctness of Improved-CBC is similar to that for CBC, and is included in Appendix A.2 for completeness. We compute the communication complexity of Improved-CBC in a similar way as in CBC:

In Lines 1(a) to 1(c), every node receives at most $n-1$ symbols, so at most $n(n-1)$ symbols are communicated in the Matching stage. With an appropriate choice of D, the complexity of Algorithm 2.6.2 can be made equal to

$$\frac{n(n-1)}{n-f}L + O(n^4 L^{0.5}). \tag{2.6}$$

So for sufficiently large $L = \Omega(n^6)$, the complexity approaches $\frac{n(n-1)}{n-f}$.

While the Improved-CBC algorithm may not be better in terms the *order* of the communication complexity, in practice, a factor of 2 reduction in communication overhead is quite significant. Whether this algorithm is optimal for arbitrary f and n ($f < n/3$) remains an open question. What we do know, however, is that the degenerate version of the algorithm for $f = 0$ with complexity $(n-1)L$ is not optimal for all n. When $f = 0$, the consensus problem reduces to the multiparty equality (MEQ) problem in which all nodes are necessarily fault-free. The MEQ problem is discussed in Chapter 4.

2.6.2 Improving the Communication Complexity for v-Validity

In Section 2.5.5, we have discussed how to achieve v-validity with CBC and analyzed its communication complexity. One weakness of the CBC algorithm is that it achieves v-validity with $O(nL)$ communication complexity only when $v = f + \Omega(n)$. In particular, for $v = f + 1$, the communication complexity of CBC becomes $O(n^2 L)$. In this section, we present VCBC, an improved algorithm that achieves v-validity with $O(nL)$ communication complexity for all all $f + 1 \leq v \leq n - f$ as long as $v = \Omega(n)$.

In VCBC, an (n, v) distance-$(n - v + 1)$ Reed-Solomon code, denoted as C_v, is used to encode v data symbols into n coded symbols. The operations in each generation g are presented in Algorithm 2.6.3

Algorithm 2.6.3 VCBC Algorithm (generation g)

In the following steps, for every node i: $R_i[k] \leftarrow S_j[k]$ whenever node i receives $S_j[k]$ from its trusted node j. If a trusted node j does not send anything when it should send $R_j[k]$, then $R_i[k] \leftarrow 0$.

1.**Matching Stage:**

Every node i performs steps 1(a) to 1(e) as follows:

(a)Compute $(S_i[1], \ldots, S_i[n]) = C_v(x_i(g))$, and *send* $S_i[i]$ to every trusted node j.

(b)If $S_i[j] = R_i[j]$ then $M_i[j] \leftarrow$ **TRUE**; else $M_i[j] \leftarrow$ **FALSE**

(c)Node i broadcasts the vector M_i using `Broadcast_Binary`

Using the received M vectors:

(d)Find a set of nodes P_{match} of size v such that

$$M_j[k] = M_k[j] = \textbf{TRUE}$$

for every pair of nodes $j, k \in P_{match}$. If multiple possibilities exist for P_{match}, then any one of the possible sets is chosen arbitrarily as P_{match} (all fault-free nodes choose a deterministic algorithm to select identical P_{match}).

(e)If P_{match} does not exist, then decide on a default value and continue to the next generation;
else continue to the following steps.

Note: At this point, if P_{match} does not exist, it is, in fact, safe to terminate the algorithm with a default output since it can be asserted that no v fault-free nodes have identical inputs. However, by continuing to the next generation instead of terminating, v-validity is satisfied for the inputs of each individual generation.

When P_{match} of size v is found, each node $i \in P_{match}$ performs steps 1(f) as follows:

(f)For every node $j \notin P_{match}$ that is trusted node i:
If $i = \min\{l | l \in P_{match}$ and node j trusts node $l\}$, then node i sends $S_i[k]$ to node j for each $k \in P_{match}$ such that node j does not trust node k.

Each node $j \notin P_{match}$ performs step 1(g) as follows:

(g)Using the first v symbols it has received from the nodes in P_{match} in steps 1(a) and 1(f), node j computes $S_j[j]$ according to C_v, then sends $S_j[j]$ to all trusted nodes.

Note: For every node i trusted by node j, it has set $R_i[j]$ to the $S_j[j]$ received from node j in step 1(a). It will be replaced with the new $S_j[j]$ received in step 1(g).

2.**Checking Stage:**

Each node i (in P_{match} or not) performs Checking Stage as follows:

(a)If $R_i \in C_v$ then $Detected_i \leftarrow$ **FALSE**; else $Detected_i \leftarrow$ **TRUE**.

(b)If node $i \in P_{match}$ and $R_i \neq S_i$ then $Detected_i \leftarrow$ **TRUE**.

(c)Broadcast $Detected_i$ using `Broadcast_Binary`.

(d)Receive $Detected_j$ from each node j (broadcast in step 2(c)). If $Detected_j =$ **FALSE** for all j, then decide on $y_i(g) = C_v^{-1}(R_i)$; else enter Diagnosis Stage.

3.**Diagnosis Stage:**

Each node i (in P_{match} or not) performs Diagnosis Stage as follows:

(a)Broadcast S_i and R_i using `Broadcast_Binary`.

(b)$S_j^{\#} \leftarrow S_j$ and $R_j^{\#} \leftarrow R_j$ received from node j as a result of broadcast in step 3(a).

Using the broadcast information, all nodes perform the following steps identically:

(c)For each edge (i, j) in Diag_Graph: Remove edge (i, j) if $\exists k$, such that node j receives $S_i[k]$ from node i in Matching stage and $R_j^{\#}[k] \neq S_i^{\#}[k]$.

(d)For each node $i \in P_{match}$: If $S_i^{\#} \notin C_v$, then node i must be faulty. So remove vertex i and the adjacent edges from Diag_Graph.

(e)For each node $j \notin P_{match}$: If $S_j^{\#}[j]$ is not consistent with the subset of v symbols of $R_j^{\#}|P_{match}$, from which $S_j^{\#}[j]$ is computed, node j must be faulty. So remove vertex j and the adjacent edges from Diag_Graph.

(f)If at least $f+1$ edges at any vertex i have been removed, then node i must be faulty. So remove vertex i and the adjacent edges.

(g)Find a set P_{decide} of maximum size such that all nodes in P_{decide} trust each other and $S_i^{\#} = S_j^{\#}$ for every pair of nodes $i, j \in P_{decide}$. In case of a tie, pick any one.

(h)If $|P_{decide}| < v$, decide on the default output.
Else, decide on $y_i(g) = C_v^{-1}(S_j^{\#})$ for any node $j \in P_{decide}$.

The proof of correctness of VCBC is similar to those for CBC and Improved-CBC, and is included in Appendix A.3 for completeness. We compute the communication complexity of VCBC in a similar way as in CBC and Improved-CBC:

In Lines 1(a) and 1(f), every node receives at most $n-1$ symbols, so at most $n(n-1)$ symbols are communicated in these two steps. In Line 1(g), every node $P_j \notin P_{match}$ sends at most $n-1$ symbols, and there are at most $n-v$ nodes not in P_{match}, so at most $(n-v)(n-1)$ symbols are communicated in this step. So in total, no more than $(2n-v)(n-1)$ symbols are communicated in the Matching stage. Then with an appropriate choice of D, the complexity of Algorithm 2.6.3 can be made to

$$\frac{(2n-v)(n-1)}{v}L + O(n^{4.5}L^{0.5}). \tag{2.7}$$

So for any $v = \Omega(n)$ and $f+1 \leq v \leq n-f$, with a sufficiently large L ($\Omega(n^7)$), the communication complexity for achieving v-validity with algorithm VCBC is $O(nL)$.

2.7 CBB – A Coding-Based Byzantine Broadcast Algorithm

In this section, we present CBB, a coding-based error-free Byzantine broadcast algorithm. For comparison, we first introduce Digest, a PBFT-like algorithm for Byzantine broadcast. Both Digest and CBB have similar structure as CBC. They both divide the L-bit input value x into generations of D bits, and a diagnose graph Diag_Graph is maintained across generations.

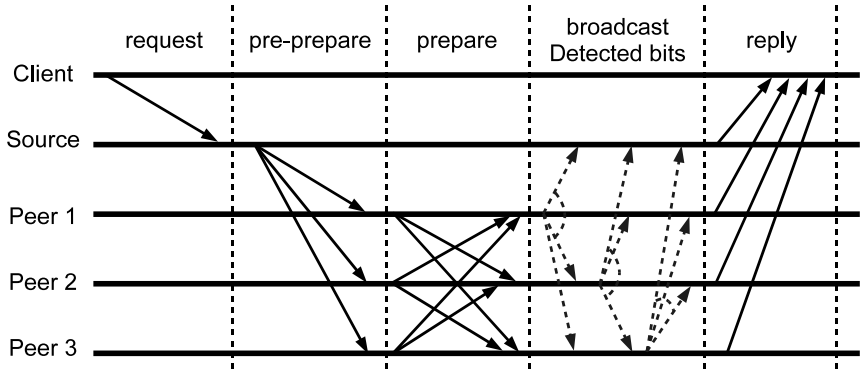


Figure 2.1: Normal case operation of Digest. Dashed arrows represent broadcasting using `Broadcast_Binary`.

2.7.1 Digest: A PBFT-Like Algorithm

In this section we briefly describe a simple PBFT-like algorithm: Digest. Digest provides the readers a general idea how algorithms such as those in [23, 25, 39, 28] utilize collision-resistant hash functions to achieve Byzantine broadcast. It will also serve as a baseline for the performance evaluation of the CBB algorithm.

In Digest, the Byzantine broadcast of the g-th generation starts with the source node n multicasting a *pre-prepare* message $\langle \texttt{PRE} - \texttt{PREPARE}, x(g) \rangle$ to all of the peers, where $x(g)$ is the input for the g-th generation.

After a peer i ($i < n$) receives pre-prepare message $\langle \texttt{PRE} - \texttt{PREPARE}, x_i(g) \rangle$ from the source node n, it sends one *prepare* message $\langle \texttt{PREPARE}, k_{i,j}, d_{i,j} \rangle$ to each peer j ($j < n$), where $k_{i,j}$ is a randomly generated key, and $d_{i,j} = H(x_i(g), k_{i,j})$ is $x_i(g)$'s "digest" computed using key $k_{i,j}$ and a pre-determined collision-resistant hash function H.

A peer i waits until it receives all $n-2$ prepare messages from the other peers. Then it checks if $d_{j,i} = H(x_i(g), k_{j,i})$ for all j. If yes, then node i sets $Detected_i$ to **FALSE**. Otherwise, it sets $Detected_i$ to **TRUE**. Then node i broadcasts an one-bit message $\langle Detected_i \rangle$ to the rest of the network (including the source node n), using `Broadcast_Binary`. Since `Broadcast_Binary` is error-free, all fault-free nodes receive identical $Detected_i$ from each peer i.

When all $n-1$ instances of `Broadcast_Binary` terminate, every node (including the source node n) checks if all $Detected_j$'s are **FALSE**. If yes, it agrees on its local copy of $x(g)$ ($x_i(g)$ if the node is a peer i). Otherwise, failure is detected and diagnosis will be performed. Figure 2.1 illustrates the

failure-free operations of Digest.

Failure-Case Operation

It should not be hard to see that whenever at least one of the $Detected_i$ is **TRUE**, the faulty node(s) must have misbehaved in one or some combination of the following ways: (1) a faulty source node sends different $x(g)$ to different peers; (2) a faulty peer i sends incorrect prepare message to one or more peers; or (3) a faulty peer i broadcasts $Detected_i$ =**TRUE** even though it should be **FALSE**. Different algorithms handle failures differently. Here we present one technique known as "dispute control" [37], which is similar to the diagnosis stage of the CBC algorithm from Section 2.5.

In dispute control, when failure is detected, every node broadcasts all messages it has sent and received in the current generation, using `Broadcast_Binary`. The peers first agree on the value $x(g)$ broadcast by the source using `Broadcast_Binary`. Then, by comparing the information broadcast by each pair of nodes, the fault-free nodes will be able to jointly update the diagnosis graph Diag_Graph identically. By the end of dispute control, at least one of the following claims is true.

1. One *new* node is correctly identified as faulty and this node has not been identified as faulty before. This node is then isolated from the rest of the system, by having all edges attached to its corresponding vertex in Diag_Graph removed. This is similar to `Line 3(f)` in Algorithm CBC.

2. One *new* pair of nodes, say nodes a, b, are "in dispute" with each other, i.e., their broadcasts, using `Broadcast_Binary`, contradict each other. When a node pair a, b is found *in dispute*, it is guaranteed that (i) *at least* one of these two nodes is faulty, and (ii) these two nodes trust each other at the beginning of the current generation. Then the edge connecting the corresponding vertices in Diag_Graph is removed. This is similar to `Line 3(e)` in Algorithm CBC.

The correctness of the above claims follows from a similar argument as those for `Lines 3(e)-3(f)` in Algorithm CBC. Our network-aware Byzantine broadcast algorithm NAB (presented in Chapter 3) also uses a similar dispute control structure with a few additional steps, which is discussed in

detail in Appendix B.2. The discussion of steps DC1-DC3 in Appendix B.2 can serve as a proof of correctness for two claims above, by substituting $\mathcal{V}_k$ as Diag_Graph.

Following the same argument for `Line 3(g)` in Algorithm CBC, any node that is accused by at least $f+1$ other nodes must be faulty, and it will be isolated, in the same way as `Line 3(g)` in Algorithm CBC. It then follows that dispute control will be performed for at most $f(f+1)$ times. If at any time the source node n is identified as faulty, all fault-free nodes terminate the algorithm and decide on a default output.

Correctness and Traffic Load Analysis

The correctness of Digest follows from the correctness of PBFT [23]. Now we consider the communication complexity of Digest in the failure-free case, because this is the scenario that determines the performance of the system [23]. Also similar to our earlier discussion of the communication complexity of CBC, the failure-free case dominates the complexity of the algorithm for sufficiently large value of L. Recall that D is the size of each generation. Let κ be the size of the key-digest pair $(k_{i,j}, d_{i,j})$ and $B = \Theta(n^2)$ be the communication cost of each execution of `Broadcast_Binary`. The per-generation complexity imposed by Digest is

$$(n-1)D + (n-1)(n-2)\kappa + (n-1)B \tag{2.8}$$

$$= (n-1)D + \kappa O(n^2) + O(n^3). \tag{2.9}$$

So for sufficiently large D ($D = \Omega(\kappa n^2)$ and $D = \Omega(n^3)$), the per-generation of Digest is roughly $(n-1)D$.

2.7.2 The CBB Algorithm

CBB has a similar structure as Digest. But instead of using hashing, it uses the technique of error-detection coding, similar to CBC. The pseudo-code of CBB is presented in Algorithm 2.7.4.

Algorithm 2.7.4 CBB Algorithm (generation g)

In the following steps, R_i is a $2(n-1)$-dimension vector, which is initialized as all $\perp$. For every peer i: $R_i[k] \leftarrow R_j[k]$ whenever node i receives $R_j[k]$ from a trusted node j. If a trusted node j does not send anything when it should send $R_j[k]$ or S_k, then $R_i[k] \leftarrow 0$.

Source node n:

1.Encode $x(g)$ into $(S_1, \cdots, S_{2(n-1)}) = C_{n-f}(x(g))$.

2.For each trusted peer i ($i < n$): Send $(S_i, S_{i+(n-1)})$ to node i.

Each peer i ($i < n$) trusted by source node n:

3.$(R_i[i], R_i[i + (n-1)]) \leftarrow (S_i, S_{i+(n-1)})$ received from node n in step 2.

4.Send $R_i[i]$ to all trusted peers.

Each peer i ($i < n$) not trusted by source node n:

5.If $k < n - f$ peers are trusted by both node i and source node n, node i receives $< n - f$ symbols in step 4:

Receive $R_j[j + (n-1)]$ from each peer j trusted by both node i and source node n, until R_i has $n - f$ non-null symbols.

6.Now R_i has $n - f$ non-null symbols:

Decode $Z = C_{n-f}^{-1}(R_i)$ and set $R_i[i]$ to be the i-th symbol of Z. If the decoding fails, set $R_i[i]$ to 0.

7.Send $R_i[i]$ to all trusted peers.

Every peer i:

8.If $R_i \notin C_{n-f}$, then $Detected_i \leftarrow$ **TRUE**;

Otherwise $Detected_i \leftarrow$ **FALSE**.

9.Broadcast $Detected_i$ using `Broadcast_Binary`.

10.Receive $Detected_j$ from each node j (broadcast in step 9):
If $Detected_j =$ **FALSE** for all j, agree on $C_{n-f}^{-1}(R_i)$.

Otherwise perform dispute control (Similar to Digest).

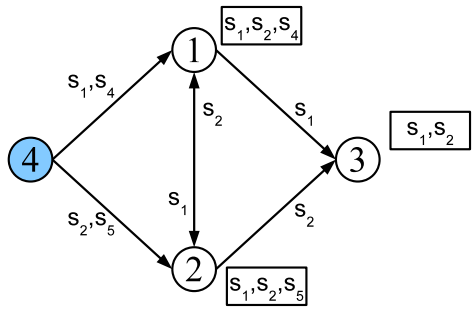
11.Proceed to the next generation.

`Lines 1 to 3:` These steps correspond to the source node n multicasting the pre-prepare message to the peers in Digest. The difference lies in that, in CBB, the source node n encodes the input of the current generation $x(g)$ into $2(n-1)$ coded symbols with code C_{n-f} and only sends a small set of the coded symbols to each of the trusted peers, rather than sending the whole D bits of $x(g)$ to every peer as in Digest. In particular, two symbols $(S_i, S_{i+(n-1)})$ are sent to every peer i that the source node n trusts.

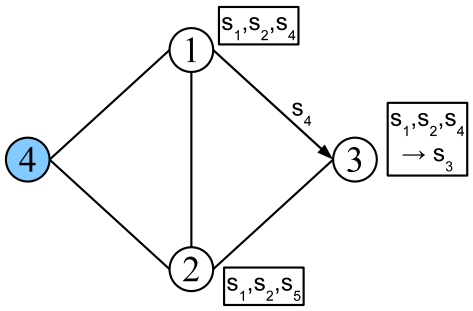
`Lines 4 to 7:` These steps correspond to the peers exchanging prepare messages in Digest. Each peer i trusted by the source node n simply relays S_i to all of its trusted peers after it is received from source node n (Figure 2.2(a)).

For a peer i not trusted by source node n, if there is any, it first gathers enough ($\geq n-f$) coded symbols from the peers trusted by both itself and the source node n (Figure 2.2(b)). Notice that both of nodes n and i must each trust at least $n-f-1$ other peers; otherwise, at least one of them is accused by at least $f+1$ other nodes, and should have already been identified as faulty and isolated from the rest of the system. Through simple counting, there must be at least $n-2f$ peers that are trusted by both nodes n and i. Given that $n \geq 3f+1$, these peers together receive at least $2(n-2f) \geq n-f+1$ coded symbols from the source node n. So it is guaranteed that peer i will receive enough coded symbols in `line 5`, which are then used to generate the i-th symbol of the codeword that peer i should have received from source node n if they had trusted each other. Then node i sends the i-th symbol to all its trusted peers (Figure 2.2(c)).

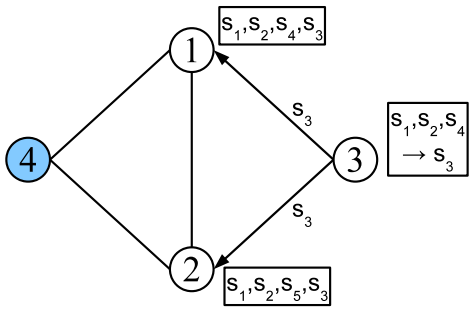
`Lines 8 and 9:` Every fault-free peer i checks whether the set of symbols received from its trusted node belong to part of a valid codeword of C_{n-f} by trying to apply the decoding function $C_{n-f}^{-1}(R_i)$. If the decoding fails, i.e., $R_i \notin C_{n-f}$, node i detects a failure/misbehavior. Then node i broadcasts $Detected_i$ using `Broadcast_Binary`. This is the counterpart of checking m_i against the received hash value and key pairs in Digest.



(a) Source node 4 does not trust node 3. Transmissions by nodes 4, 1 and 2 are shown. Node 3 receives $2 < n - f = 3$ coded symbols by the end of step 4.



(b) Peer 3 receives S_4 from node 1 so that it has $3 = n - f$ coded symbols. Then node 3 uses them to reconstruct S_3 in step 6.



(c) Peer 3 sends S_3 to all its trusted peers. By the end of step 7, all peers share packets S_1, S_2, S_3.

Figure 2.2: Example: One peer does not trust the source with $n = 4$ and $f = 1$. Two nodes connected by an edge in the figure trust each other. The direction of an arrow indicates the direction of a transmission along that edge. Next to each arrow, the coded symbol(s) transmitted on that edge is listed. The boxes near to the peers indicate the coded symbols that are available to the peers by the end of steps 4, 6 and 7, respectively.

`Line 10:` If no peer claims that it detects a failure, then each peer i agrees on $C_{n-f}^{-1}(R_i)$ as the output of the current generation. Otherwise, dispute control is performed in the same way as in Digest.

2.7.3 Correctness and Communication Complexity of CBB

The correctness of CBB is guaranteed by Theorem 2.2. The proof is similar to that for Lemma 2.3. The proof is included in Appendix A.4 for completeness.

Theorem 2.2 *If none of the peers claims failure detected, the peers agree on an identical output y. In the case the source node n is fault-free, $y = x$.*

The communication complexity of CBB, denoted as C_{CBB}, is computed similarly to that of CBC. In each generation

- `Lines 1-7`: every node i receives at most n symbols, each of $D/(n-f)$ bits, from the nodes that it trusts. So at most $\frac{n(n-1)}{n-f}D$ bits in total are transmitted by all n nodes.
- `Lines 8-9`: every peer i that is not identified as faulty broadcasts one bit $Detected_i$ with `Broadcast_Binary`. So at most $(n-1)B$ bits are transmitted.
- Dispute control: every symbol transmitted in Lines 1-7 is broadcast by two nodes with `Broadcast_Binary`. So the complexity is $\frac{2n(n-1)}{n-f}DB$ bits.

Recall that there are L/D generations in total, and dispute control is performed by at most $f(f+1)$ times. So the total communication complexity of CBB can be formulated as

$$\begin{aligned} C_{CBB}(L) &= \frac{n(n-1)}{n-f}L + (n-1)B\frac{L}{D} + f(f+1)\frac{2n(n-1)}{n-2f}DB \\ &= \frac{n(n-1)}{n-f}L + O(n^4L^{0.5}), \quad if \quad D = \Theta(L^{0.5}/n). \end{aligned} \tag{2.10}$$

Similar to $\overline{C}_{CBC}(g, D)$, we define the average per-generation complexity for performing g instances of Byzantine broadcast on D-bit values as $\overline{C}_{CBB}(g, D)$, which is formulated as

$$\begin{aligned} \overline{C}_{CBB}(g, D) &= \frac{n(n-1)}{n-f}D + (n-1)B + \frac{f(f+1)}{g}\frac{2n(n-1)}{n-2f}DB \\ &= O\left(\left(n + \frac{n^5}{g}\right)D + n^3\right). \end{aligned} \tag{2.11}$$

So for $D = \Omega(n^2)$ and $g = \Omega(n^5)$, the per-generation communication complexity imposed by CBB is roughly $\frac{n(n-1)}{n-f}D < 1.5(n-1)D$, since $f < n/3$. Recall that the cost of Digest is approximately $(n-1)D$. So in terms of traffic load, CBB is *at most* 50% more expensive than Digest. But the decoding function $C_{n-f}^{-1}()$ is much less expensive in computational complexity for small f, compared to the hash function used in Digest. The performance of the algorithms is determined by the combination of communication and

computational costs. As we will see later, according to experimental results, CBB performs comparably with Digest, without the need for a secure hash function.

2.7.4 Implementation

For performance evaluation, we implemented both Digest and CBB in Linux. In this section, we discuss some of our implementation choices and some optimization we adopted for CBB.

MultiThreading

We implement both Digest and CBB in a multithreading fashion. On every node, there is one listener thread for each of the other nodes, which simply accepts incoming connection requests from the node that it listens to. TCP is used for communication between every pair of nodes. Also since Digest and CBB share the same structure, it also makes it easier to share the same code between the two algorithms.

A worker thread is launched whenever a listener thread receives a TCP connection request. The worker thread establishes a TCP session with the requesting node. When the TCP session completes, the worker thread checks whether the received information follows the specification of the algorithm, and if enough messages have been received from different nodes. If yes, it proceeds to the consistency checking part (checking the digests against the local generation in Digest, and decoding $C^{-1}(R_i)$ in CBB) and the rest of the algorithm.

Of course, the algorithms can be implemented in the sequential manner as well. However, this will require careful coordination among the nodes about when and who should transmit to whom. Otherwise the system will end up deadlocked when two nodes try to send messages to each other. This can be very complicated in large systems. With multithreading, this complication is avoided.

Multithreading also makes it very easy to incorporate different optimizations. For example, with "speculative execution" technique in Zyzzyva [24], a node executes the request speculatively when it receives a minimum number of matching prepare messages, even if it cannot confirm that all other nodes

have accepted the same request. So by the time it confirms an agreement of the request, the output has already been computed and is waiting to be sent to the client. This technique can be incorporated into our implementation by having the worker thread launch a "speculate" thread for performing the request speculatively.

Synchrony

Although we assume a synchronous system model in the previous discussion, bad things can happen in real life. Synchrony might be violated temporarily in practice. For example, packets might be lost in the network and never reach the destination. Since we use TCP as the underlying communication algorithm, packet losses in the middle of a TCP session have been taken care of by the retransmission mechanism of TCP. So the only problem packet losses can cause is when one node attempts to create a connection with another node. Our implementation incorporates this concern by allowing each node to try to connect to another node for no more than `MAX_ATTEMPTS` times. If a node fails to create a connection with another node after `MAX_ATTEMPTS` attempts, it considers that node as being faulty. In our experiments, `MAX_ATTEMPTS` is set to 2.

Optimization for single failure

In replication systems where each individual replica is reasonably reliable, the probability that more than one node fails at the same time is very small. In this case, designing for $f = 1$ suffices, and CBB can be optimized to further reduce the computational cost as follows.

Instead of using a $(2(n-1), n-1)$ Reed-Solomon code, we use an $(n, n-1)$ Reed-Solomon code, which is in fact a parity code: $S_1, \cdots, S_{n-1}$ are just raw data symbols, and the parity symbol $S_n = \oplus_{i=1}^{n-1} S_i$, where $\oplus$ denotes bitwise exclusive or (XOR). Then we simply substitute all $S_{i+(n-1)}$ in Algorithm 2.7.4 with S_n. For failure detection, a peer just needs to check if the received symbols pass the parity check. If all peers claim with `Broadcast_Binary` that their symbols passed the parity check, then the output is simply $S_1, \cdots, S_{n-1}$. The computation for this modified CBB algorithm is very efficient: each node just needs to perform bitwise XOR's.

2.7.5 Experimental Evaluation

We dedicate this section to investigations of how Byzantine broadcast algorithms perform in real systems. We implemented four algorithms in Linux: Digest, CBB and the following two:

- BASIC: This is the classic algorithm developed by Pease, Shostak and Lamport [3]. It is also used to realize `Broadcast_Binary` in the other algorithms. When there is at most one faulty node ($f = 1$), BASIC is very simple: the source sends the value x to every peer and the peer forwards that to every other peer. The output at each peer is computed as the majority of the received values. If there is no clear majority, then all peers know that source must be faulty and will agree on some default message.

- BTH: The error-free Byzantine broadcast algorithm proposed by Beerliova-Trubiniova and Hirt in [31]. This algorithm uses similar ideas of coding and fault diagnosis as CBB to reduce communication complexity to $O(nL)$. But it has a higher communication and computational cost than CBB.

We conduct our experiments in systems of both four and five nodes: four (or five) Dell Inspiron 1545 laptops connected by a Netgear GS108 gigabit switch. Each Inspiron 1545 was running Ubuntu 9.04 and a 2.0 GHz Intel Core 2 Duo Processor T6400. One Inspiron machine was designated to be the source node and the remaining three (or four) machines made up the group of peers. In order to ensure consistency between test runs, the role performed by each machine remained constant throughout the experiment. In Digest, we use SHA-256 as the collision-resistant hash function. We use the realization of SHA-256 from the OpenSSL toolkit [48]. The results with five nodes are very similar to the ones with four nodes. So we only present results for four nodes in this section.

To minimize any external error introduced by thread scheduling and background services, each data point in the following discussion/figures represents an average of 100 identical trials. To further reduce any skew, the trials between the three different algorithms were interleaved so that any unforeseeable incident would affect the algorithms as evenly as possible. In each trial, the actual content of each generation was randomly generated. This

random generation occurred at run time and differed for each algorithm. We see no reason for the content of each generation to have any bearing on the results of our experiment. For each set of 100 trials, the generation size D and the number of generations g is fixed. All algorithms were timed from the moment the source begins executing at the first generation until the moment that broadcast of all g generations terminates at the source node. Timing only the source is an accurate measure since for every algorithm the source cannot terminate until it has reliably received all $n-1$ $Decided_i$ bits as **FALSE** from `Broadcast_Binary`. So when the source terminates, it is guaranteed that all peers have agreed upon the correct information.

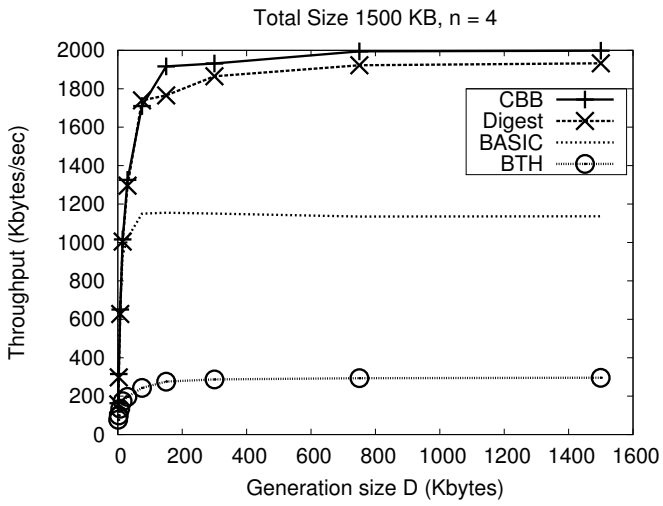
Effects of Generation Sizes

We first test the failure-free performance of CBB, Digest, BASIC and BTH for different values of generation size D. In particular, we measure each algorithm's throughput: total size of data in bytes divided by the time it takes to finish broadcasting. For the experiments of four nodes ($n = 4$ and $f = 1$), the per-generation communication costs of the four implemented algorithms for large D are approximately,

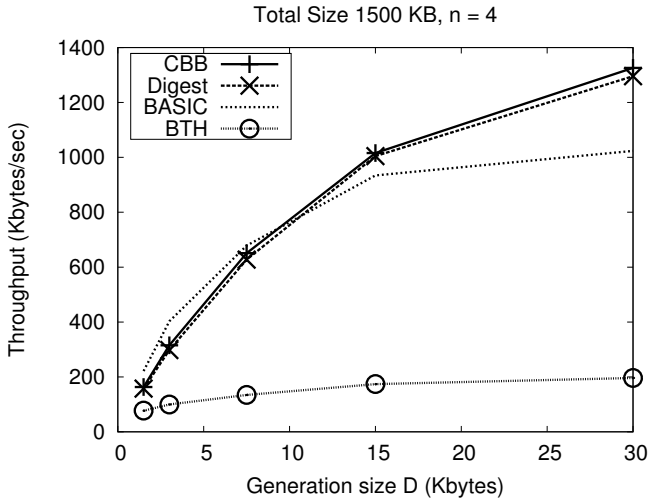
- CBB: $n(n-1)D/(n-f) = 4D$;
- Digest: $(n-1)D = 3D$;
- BASIC: $9D$;
- BTH: $2n(n-1)D/(n-2f) = 12D$.

It follows that, if only communication cost is considered, the throughput of Digest should be 4/3 times that of CBB, while the throughput of BASIC and BTH should be 4/9 and 1/3 times the throughput of CBB, respectively.

Figure 2.3(a) shows how the throughputs of the four tested algorithms change as the generation size D varies from 1.5KB to 1500KB, while the total size L is fixed to 1500KB, and Figure 2.3(b) zooms-in for generation size $D \leq$ 30KB. We first observe that CBB and Digest achieve similar throughput for all values of D. For all values of D, the throughput of CBB is at least as high as that of Digest, and for $D \geq$ 150KB, the throughput of CBB is even higher than that of Digest by roughly $3\% \sim 8\%$. This justifies our earlier



(a) Generation size $\leq$ 1500 KB



(b) Generation size $\leq$ 30 KB

Figure 2.3: Throughput under different generation sizes.

argument that although CBB introduces more traffic than Digest does, it can still outperform Digest in practice, since CBB does not need to compute any collision-resistant hash function, resulting in a lower computational cost than Digest.

We next observe that the throughput of BASIC is roughly 60% that of CBB for $D \geq$ 150KB. This is slightly higher than we expect according to the comparison of these two algorithms' per-generation communication cost. We believe this is the joint effect of the extra communication overhead in CBB introduced by the executions of `Broadcast_Binary` for broadcast-

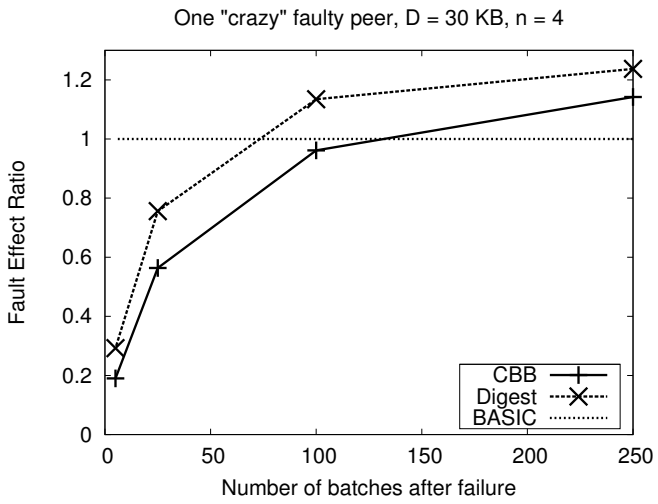


Figure 2.4: One "crazy" faulty peer keeps sending arbitrary corrupted messages to all other nodes, starting from the first generation.

ing the *Detected* bits and the computational overhead introduced by encoding/decoding for the Reed-Solomon code C_{n-f}. Finally, BTH achieves roughly 15% of CBB's throughput. This is about 1/2 lower than we expect according to the previous analysis. We believe this is mainly because that BTH uses a more complicated code than CBB, which results in higher computational cost and hence reduces its throughput, and includes an additional round of messages exchange than CBB.

Performance with Failure

We also conduct tests for the failure cases. To quantify the effect of the presence of one faulty node on the performance of the algorithms, we define *"fault effect ratio"* as:

$$\frac{Throughput\ after\ failure}{Throughput\ before\ failure}. \tag{2.12}$$

It reflects how much slower/faster an algorithm becomes when one of the nodes goes "bad" (or fails). At this time, we have only implemented the normal case of BTH. So we will only compare CBB, Digest and BASIC below.

We first test the scenario when a faulty peer goes "crazy": the faulty peer sends arbitrary corrupted messages to all other nodes. Figure 2.4 shows how

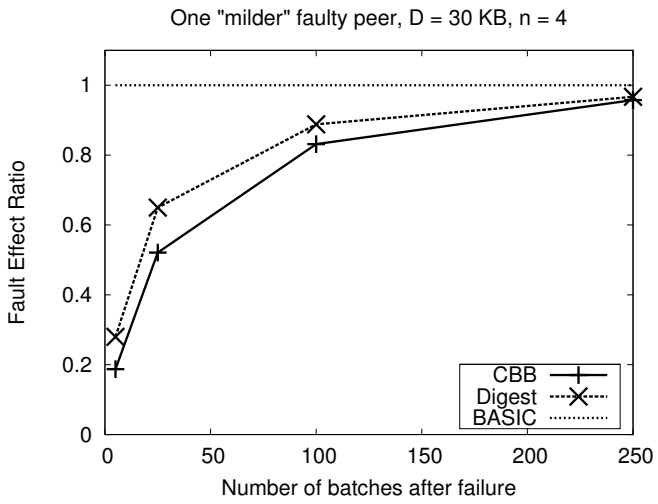


Figure 2.5: One "milder" faulty peer keeps sending arbitrary corrupted messages to only one other node, starting from the first generation.

the fault effect ratios of CBB, Digest and BASIC change after the faulty peer becomes "crazy". It is easy to see that BASIC's performance does not depend on the presence of failures, so its fault effect ratio is always 1. On the other hand, immediately after the faulty peer becomes "crazy", the fault effect ratio is as small as 0.2 for CBB and about 0.3 for Digest. This means when a faulty node starts to misbehave, throughputs of CBB and Digest are reduced to approximately 20% and 30% of their throughput before failure, respectively. The reduction is due to the fault diagnosis process, in which every node is required to broadcast everything it has received or sent using `Broadcast_Binary`. As the number of generations after failure increases, the fault effect ratio increases. After 250 generations, the fault effect ratio becomes larger than 1 (roughly 1.15 - 1.25), which implies that when a faulty node misbehaves, the system, in fact, becomes faster in the long run. It may seem counter-intuitive that a faulty node could improve the performance. The explanation for this phenomenon is that since the faulty peer sends corrupted messages to all other nodes, conflicts are found between the faulty peer and all other nodes during the fault diagnosis process. Then the faulty peer is immediately identified by the others. Hence all fault-free nodes will not communicate with the faulty peer, and do not need to check the consistency of the faulty peer's messages in the subsequent generations. As a result, both communication and computation costs are reduced after the

first generation, and hence the algorithms become more efficient. The cost of diagnosis in the first generation becomes negligible once it is amortized over a large number of generations. Since the computational cost of the hash function in Digest is much higher than the decoding function of CBB, Digest saves more than CBB from identifying the faulty node. As a result, the fault effect ratio of Digest is always slightly larger than that of CBB.

We also test a "milder" faulty peer, which only sends corrupted messages to one fault-free peer. In this scenario, only one conflict is found and the faulty node is not exactly identified. In Figure 2.5, a similar trend as in Figure 2.4 is observed. R converges to 1 when the number of generations is large, which means that the performance is the same as in normal-case, because the faulty node is not isolated.

Both tests show that with the fault diagnosis process in Digest and CBB, the misbehavior of faulty nodes will only degrade the performance of the system temporarily, and the long term performance will not be degraded (or may even be improved), even if the faulty nodes persistently misbehave.

2.8 Summary

In this chapter, we have presented two efficient error-free Byzantine fault-tolerant algorithms: CBC for the consensus problem, and CBB for the broadcast problem. Both algorithms require $O(nL)$ total bits of communication for achieving Byzantine consensus and Byzantine broadcast, respectively, of L bits for sufficiently large L. These algorithms make no cryptographic assumption.

CHAPTER 3

NETWORK-AWARE BYZANTINE AGREEMENT ALGORITHM DESIGN

3.1 Introduction

In Chapter 2, we present a Byzantine consensus algorithm CBC and a Byzantine broadcast algorithm CBB that solve the consensus and broadcast problems with communication complexity $O(nL)$ for sufficiently large L. Both CBC and CBB, as well as most existing Byzantine consensus/broadcast algorithms, are *network-oblivious*, by which we mean that these algorithms are designed without taking into account constraints of the underlying communication networks.

In this chapter, we study the design of *Network-Aware* Byzantine broadcast algorithms. The proposed Byzantine broadcast algorithms are "network-aware" in the sense that their design takes the link capacities into account. In particular, we consider the problem of maximizing the *throughput* of Byzantine broadcast in *synchronous* networks of point-to-point links, wherein each directed communication link is subject to a "capacity" constraint. Informally speaking, *throughput* of Byzantine broadcast is the number of bits that can be broadcast from the source to the peers per unit time (on average), under the worst-case behavior by the faulty nodes. Despite the large body of work on Byzantine consensus and broadcast [49, 35, 34, 36, 31, 38], performance of consensus and broadcast in *arbitrary* point-to-point network has not been investigated previously. When capacities of the different links are not identical, previously proposed algorithms can perform poorly. In fact, one can easily construct example networks in which previously proposed algorithms achieve throughput that is arbitrarily worse than the optimal throughput.

Main contributions: This chapter studies throughput and capacity of Byzantine broadcast in point-to-point networks.

1. We develop a Network-Aware Byzantine broadcast algorithm NAB for *arbitrary* point-to-point networks wherein each directed communication link is subject to a capacity constraint.

2. We derive an upper bound on the capacity of BB in arbitrary point-to-point networks.

3. We show that NAB can achieve throughput at least 1/3 of the capacity in arbitrary point-to-point networks. When the network satisfies an additional condition, NAB can achieve throughput at least 1/2 of the capacity.

4. We develop Capacity-Achieving network-aware Byzantine broadcast (CAB) algorithms for two special families of networks, namely 4-node networks and symmetric fully connected networks.

This chapter is structured as follows. In Section 3.2, we first recapitulate the definition of the Byzantine broadcast problem, and introduce the formal definition of the point-to-point network. We then define throughput and capacity of Byzantine broadcast in a given network. Overview of the NAB algorithm is then presented in Section 3.3. Detailed discussion of the NAB algorithm is given in Sections 3.4 and 3.6. A general upper bound on the capacity of Byzantine broadcast in arbitrary point-to-point networks is then proved in Section 3.6.2. Then in Section 3.7 we introduce the CAB algorithm, which can achieve a throughput arbitrarily close to the upper bound in 4-nodes networks and symmetric fully-connected networks.

3.2 Problem Definition and Models

We have introduced the definition of the Byzantine broadcast problem in Chapter 2.1. For convenience in the following discussion, we restate the definition below. We consider a *synchronous* system consisting of n nodes, named $1, 2, \cdots, n$, with one node designated as the *sender* or *source* node. In particular, we assume here that node 1 is the source node. Source node 1 is given an *input* value x containing L bits, and the goal here is for the source to broadcast its input to all the other nodes. The following conditions must be satisfied when the input value at the source node is x:

- **Termination:** Every fault-free node i must eventually decide on an *output* value of L bits; let us denote the output value of fault-free node i as y_i.

- **Agreement:** All fault-free nodes must agree on an identical output value, i.e., there exists y such that $y_i = y$ for each fault-free node i.

- **Validity:** If the source node is fault-free, then the agreed value must be identical to the input value of the source, i.e., $y = x$.

Failure Model: The faulty nodes are controlled by an adversary that has a complete knowledge of the network topology, the algorithm, and the information the source is trying to send. No secret is hidden from the adversary. The adversary can take over at most f nodes at any point during execution of the algorithm, where $f < n/3$. These nodes are said to be *faulty*. The faulty nodes can engage in any kind of deviations from the algorithm, including sending incorrect or inconsistent messages to the neighbors.

We assume that the set of faulty nodes remains fixed across different instances of execution of the BB algorithm. This assumption captures the conditions in practical replicated server systems. In such a system, the replicas may use Byzantine broadcast to agree on requests to be processed. The set of faulty (or compromised) replicas that may adversely affect the agreement on each request does *not* change arbitrarily. We model this by assuming that the set of faulty nodes remains fixed over time.

When a faulty node fails to send a message to a neighbor as required by the algorithm, we assume that the recipient node interprets the missing message as being some default value.

Network Model: We assume a synchronous point-to-point network modeled as a directed simple graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the set of vertices $\mathcal{V} = \{1, 2, \cdots, n\}$ represents the nodes in the point-to-point network, and the set of edges $\mathcal{E}$ represents the links in the network. The *capacity* of an edge $e \in \mathcal{E}$ is denoted as z_e. With a slight abuse of terminology, we will use the terms *edge* and *link* interchangeably, and use the terms *vertex* and *node* interchangeably. We assume that $n \geq 3f + 1$ and that the network connectivity is at least $2f + 1$ (these two conditions are necessary for the existence of a correct BB algorithm [49]).

In the given network, links may not exist between all node pairs. Each directed link is associated with a *fixed* link capacity, which specifies the maximum amount of information that can be transmitted on that link per unit time. Specifically, over a directed edge $e = (i, j)$ with capacity z_e bits/unit time, we assume that up to $z_e\tau$ bits can be reliably sent from node i to node j over time duration τ (for any non-negative τ). This is a deterministic model of *capacity* that has been commonly used in other work [50, 45, 51, 46]. All link capacities are assumed to be positive integers. Rational link capacities can be turned into integers by choosing a suitable time unit. Irrational link capacities can be approximated by integers with arbitrary accuracy by choosing a suitably long time unit. Propagation delays on the links are assumed to be zero (relaxing this assumption does not impact the correctness of results shown for large input sizes). We also assume that each node correctly knows the identity of the nodes at the other end of its links.

This point-to-point communication model has been widely used for wired networks. It can also be used for wireless networks, if capacity constraint is imposed for each point-to-point wireless link.

Throughput and Capacity of Byzantine Broadcast

When defining the throughput of a given BB algorithm in a given network, we consider $Q \geq 1$ independent instances of BB. The source node is given an L-bit input for each of these Q instances, and the *validity* and *agreement* properties need to be satisfied for each instance *separately* (i.e., independent of the outcome for the other instances).

For any BB algorithm $\mathcal{A}$, denote $t(\mathcal{G}, L, Q, \mathcal{A})$ as the duration of time required, in the worst case, to complete Q instances of L-bit Byzantine broadcast, without violating the capacity constraints of the links in $\mathcal{G}$. Throughput of algorithm $\mathcal{A}$ in network $\mathcal{G}$ for L-bit inputs is then defined as

$$T(\mathcal{G}, L, \mathcal{A}) = \lim_{Q \to \infty} \frac{LQ}{t(\mathcal{G}, L, Q, \mathcal{A})}.$$

We then define capacity C_{BB} as follows.

Capacity C_{BB} of Byzantine broadcast in network $\mathcal{G}$ is defined as the supremum over the throughput of all algorithms $\mathcal{A}$ that solve the BB problem and

all values of L. That is,

$$C_{BB}(\mathcal{G}) \;=\; \sup_{\mathcal{A},L} T(\mathcal{G}, L, \mathcal{A}). \tag{3.1}$$

Generalization of the Problem

Recall that the goal of Byzantine broadcast is for *all* nodes in the system, except for the source node, to learn about the input value (if source is fault-free). In practice, similar to the multicast problem in networking, we may only need to reliably deliver the input value to a *subset* of the nodes in the system, namely the receivers; there are some other nodes in the system, namely the helpers, that are just to "help" the delivery of the input value and themselves do not necessary need to learn the input value. For example, an Ethernet switch or a router can be considered as a helper node. If this is the case, then the Byzantine Broadcast problem can be generalized in many ways. We list some possibly interesting generalizations below:

- We do not differentiate the faulty nodes: the only constraint on the capability of the adversary is that it can control up to f nodes (can include the source, the receivers, and the helpers).
- We differentiate the faulty nodes: the adversary can control up to f_1 non-helper nodes (source and receivers), and up to f_2 helpers.
- In addition to the link capacity constraints in $\mathcal{G}$, each helper node has a *total* traffic constraint. In other words, the total number of bits a helper node can send and receive together per unit time cannot exceed its total traffic constraint.

There are many more ways to generalize the problem. However, for this dissertation, we only consider the *original* problem in which all non-source nodes in the system are the receivers and the only constraints are the link capacity constraints.

3.3 Algorithm Overview

Each instance of our NAB algorithm performs Byzantine broadcast of an L-bit value. We assume that the NAB algorithm is used repeatedly, and during all these repeated executions, the cumulative number of faulty nodes is upper bounded by f. Due to this assumption, the algorithm can perform well by amortizing the cost of fault tolerance over a large number of executions. Larger values of L also result in better performance for the algorithm. The algorithm is intended to be used for sufficiently large L, to be elaborated later.

The k-th instance of NAB executes on a network corresponding to graph $\mathcal{G}_k(\mathcal{V}_k, \mathcal{E}_k)$, defined as follows:

- For the first instance, $k = 1$, and $\mathcal{G}_1 = \mathcal{G}$. Thus, $\mathcal{V}_1 = \mathcal{V}$ and $\mathcal{E}_1 = \mathcal{E}$.
- The k-th instance of NAB occurs on graph $\mathcal{G}_k$ in the following sense: (i) all the fault-free nodes know the node and edge sets $\mathcal{V}_k$ and $\mathcal{E}_k$, (ii) only the nodes corresponding to the vertices in $\mathcal{V}_k$ need to participate in the k-th instance of BB, and (iii) only the links corresponding to the edges in $\mathcal{E}_k$ are used for communication in the k-th instance of NAB (communication received on other links can be ignored).

  During the k-th instance of NAB using graph $\mathcal{G}_k$, if misbehavior by some faulty node(s) is detected, then, as described later, additional information is gleaned about the potential identity of the faulty node(s). In this case, $\mathcal{G}_{k+1}$ is obtained by removing from $\mathcal{G}_k$ appropriately chosen edges and possibly some vertices (as described later).

  On the other hand, if during the k-th instance, no misbehavior is detected, then $\mathcal{G}_{k+1} = \mathcal{G}_k$.

The k-th instance of NAB algorithm consists of three phases, as described next. The main contributions of this chapter are (i) the algorithm used in Phase 2 below, and (ii) a performance analysis of NAB.

If graph $\mathcal{G}_k$ does **not** contain the source node 1, then (as will be clearer later) by the start of the k-th instance of NAB, all the fault-free nodes already know that the source node is surely faulty; in this case, the fault-free nodes can agree on a default value for the output, and terminate the algorithm. Hereafter, we will assume that the source node 1 is in $\mathcal{G}_k$.

Phase 1: Unreliable Broadcast

In Phase 1, source node 1 broadcasts L bits to all the other nodes in $\mathcal{G}_k$. This phase makes no effort to detect or tolerate misbehavior by faulty nodes. As elaborated in Appendix B.1, unreliable broadcast can be performed using a set of spanning trees embedded in graph $\mathcal{G}_k$. Now let us analyze the time required to perform unreliable broadcast in Phase 1.

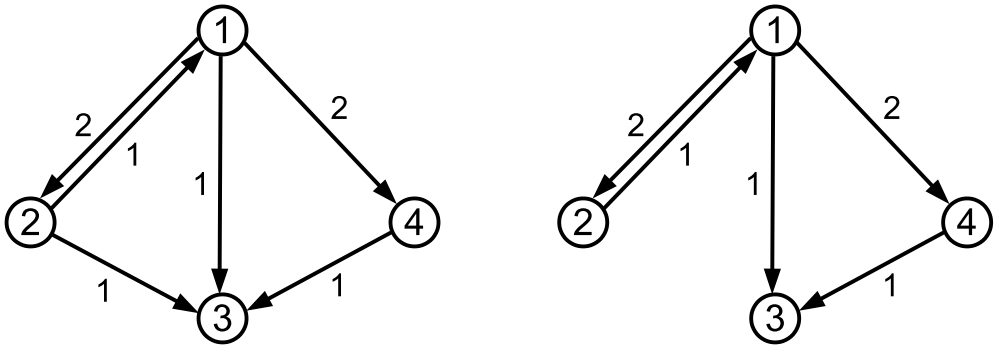
$MINCUT(\mathcal{G}_k, 1, j)$ denotes the minimum cut in the directed graph $\mathcal{G}_k$ from source node 1 to node j. Let us define

$$\gamma_k = \min_{j \in \mathcal{V}_k} MINCUT(\mathcal{G}_k, 1, j).$$

$MINCUT(\mathcal{G}_k, 1, j)$ is equal to the maximum flow rate possible from node 1 to node $j \in \mathcal{V}_k$. It is well-known [52] that γ_k is the maximum rate achievable for unreliable broadcast from node 1 to *all* the other nodes in $\mathcal{V}_k$, under the capacity constraints on the links in $\mathcal{E}_k$. Thus, the least amount of time in which L bits can be broadcast by node 1 in graph $\mathcal{G}_k$ is given by[1]

$$L \ / \ \gamma_k. \tag{3.2}$$

Clearly, γ_k depends on the capacities of the links in $\mathcal{G}_k$. For example, if $\mathcal{G}_k$ were the directed graph in Figure 3.1(a), then $MINCUT(\mathcal{G}_k, 1, 2) = MINCUT(\mathcal{G}_k, 1, 4) = 2$, $MINCUT(\mathcal{G}_k, 1, 3) = 3$, and hence $\gamma_k = 2$.



(a) (b)

Figure 3.1: Example graphs.

[1]To simplify the analysis, we ignore propagation delays. Analogous results on throughput and capacity can be obtained in the presence of propagation delays as well.

At the end of the broadcast operation in Phase 1, each node should have received L bits. At the end of Phase 1 of the k-th instance of NAB, one of the following four outcomes will occur:

(i) the source node 1 is fault-free, and all the fault-free nodes correctly receive the source node's L-bit input for the k-th instance of NAB, or

(ii) the source node 1 is fault-free, but some of the fault-free nodes receive incorrect L-bit values due to misbehavior by some faulty node(s), or

(iii) the source node 1 is faulty, but all the fault-free nodes still receive an identical L-bit value in Phase 1, or

(iv) The source node is faulty, and all the fault-free nodes do not receive an identical L-bit value in Phase 1.

The values received by the fault-free nodes in cases (i) and (iii) satisfy the *agreement* and *validity* conditions, whereas in cases (ii) and (iv) at least one of the two conditions is violated.

Phase 2: Failure Detection

Phase 2 performs the following two operations. As stipulated in the fault model, a faulty node may not follow the algorithm specification correctly.

- (Step 2.1) *Equality check:* Using an *Equality Check* algorithm, the nodes in $\mathcal{V}_k$ perform a comparison of the L-bit value they received in Phase 1, to determine if all the nodes received an identical value. The source node 1 also participates in this comparison operation (treating its input as the value "received from" itself).

  Section 3.4 presents the *Equality Check* algorithm, which is designed to **guarantee** that if the values received by the fault-free nodes in Phase 1 are **not identical**, then at least one fault-free node will detect the **mismatch**.

- (Step 2.2) *Agreeing on the outcome of equality check:* Using a previously proposed Byzantine broadcast algorithm, such as [3], each node performs Byzantine broadcast of a 1-bit *flag* to other nodes in $\mathcal{G}_k$ indicating whether it detected a mismatch during *Equality Check*.

If any node broadcasts in step 2.2 that it has detected a mismatch, then subsequently **Phase 3 is performed**. On the other hand, if no node announces a mismatch in step 2.2 above, then **Phase 3 is not performed**; in this case, each fault-free node agrees on the value it received in Phase 1, and the k-th instance of **NAB is completed**.

We will later prove that, when Phase 3 is **not** performed, the values agreed above by the fault-free nodes satisfy the *validity* and *agreement* conditions for the k-th instance of NAB. On the other hand, when Phase 3 is performed during the k-th instance of NAB, as noted below, Phase 3 results in correct outcome for the k-th instance.

When Phase 3 is performed, Phase 3 determines $\mathcal{G}_{k+1}$. Otherwise, $\mathcal{G}_{k+1} = \mathcal{G}_k$.

Phase 3: Dispute Control

Phase 3 employs a *dispute control* mechanism that has also been used in prior work [37, 5] and our design of linear-complexity Byzantine consensus and broadcast algorithms (CBC and CBB) in the previous chapter. Dispute control is used to update the communication graph $\mathcal{G}_k$ used for the each instance of NAB. The way $\mathcal{G}_k$ is updated is similar to the way in which Diag_Graph is updated in CBC and CBB. The main difference is that, in order to better utilize the link capacities in the network, some operations performed in dispute control of NAB are more complicated than in CBC and CBB. Appendix B.2 provides the details of the dispute control algorithm used in Phase 3. Here we summarize the outcomes of this phase – this summary should suffice for understanding the main contributions of this chapter.

The dispute control in Phase 3 has very high overhead, due to the large amount of data that needs to be transmitted. From the above discussion of Phase 2, it follows that Phase 3 is performed **only if** at least one faulty node misbehaves during Phases 1 or 2. The outcomes from Phase 3 performed during the k-th instance of NAB are as follows.

- Phase 3 results in correct Byzantine broadcast for the k-th instance of NAB. This is obtained as a byproduct of the Dispute Control mechanism.

- By the end of Phase 3, either one of the nodes in $\mathcal{V}_k$ is correctly iden-

tified as faulty, or/and at least one pair of nodes in $\mathcal{V}_k$, say nodes a, b, is identified as being "in dispute" with each other. When a node pair a, b is found *in dispute*, it is guaranteed that (i) *at least* one of these two nodes is faulty, and (ii) at least one of the directed edges (a, b) and (b, a) is in $\mathcal{E}_k$. Note that the dispute control phase **never** finds two fault-free nodes in dispute with each other.

- Phase 3 in the k-th instance of NAB computes graph $\mathcal{G}_{k+1}$. In particular, any nodes that can be inferred as being faulty based on their behavior so far are excluded from $\mathcal{V}_{k+1}$; links attached to such nodes are excluded from $\mathcal{E}_{k+1}$. In Appendix B.2 we elaborate on how the faulty nodes are identified. Then, for each node pair in $\mathcal{V}_{k+1}$, if that node pair has been found in dispute at least in one instance of NAB so far, the links between the node pair are exluded from $\mathcal{E}_{k+1}$. Phase 3 ensures that all the fault-free nodes compute an identical graph $\mathcal{G}_{k+1} = (\mathcal{V}_{k+1}, \mathcal{E}_{k+1})$ to be used during the next instance of NAB.

Consider two special cases for the k-th instance of NAB:

- If graph $\mathcal{G}_k$ does not contain the source node 1, it implies that all the fault-free nodes are aware that node 1 is faulty. In this case, they can safely agree on a default value as the outcome for the k-th instance of NAB.

- Similarly, if the source node is in $\mathcal{G}_k$ but f other nodes are excluded from $\mathcal{G}_k$, that implies that the remaining nodes in $\mathcal{G}_k$ are all fault-free; in this case, algorithm NAB can be reduced to just Phase 1.

Observe that during each execution of Phase 3, either a new pair of nodes *in dispute* is identified, or a new node is identified as faulty. Once a node is found to be in dispute with $f + 1$ distinct nodes, it can be identified as faulty, and excluded from the algorithm's execution. Therefore, Dispute Control needs to be performed at most $f(f + 1)$ times over repeated executions of NAB. Thus, even though each dispute control phase is expensive, the bounded number ensures that the amortized cost over a large number of instances of NAB is small, as reflected in the performance analysis of NAB (in Section 3.6 and Appendix B.4).

3.4 Equality Check Algorithm with Parameter ρ_k

We now present the *Equality Check* algorithm used in Phase 2, which has an integer parameter ρ_k for the k-th instance of NAB. Later in this section, we will elaborate on the choice of ρ_k, which is dependent on capacities of the links in $\mathcal{G}_k$.

Let us denote by x_i the L-bit value received by fault-free node $i \in \mathcal{V}_k$ in Phase 1 of the k-th instance. For simplicity, we do not include index k in the notation x_i. To simplify the presentation, let us assume that L/ρ_k is an integer. Thus we can represent the L-bit value x_i as ρ_k symbols from Galois Field $GF(2^{L/\rho_k})$. In particular, we represent x_i as a $1 \times \rho_k$ vector $\mathbf{X_i}$ in $GF(2^{L/\rho_k})$,

$$\mathbf{X_i} = [X_i(1),\, X_i(2), \cdots, X_i(\rho_k)]$$

where each symbol $X_i(j) \in GF(2^{L/\rho_k})$ can be represented using L/ρ_k bits. As discussed earlier, for convenience, we assume that all the link capacities are integers when using a suitable time unit.

Algorithm 3.4.5 Equality Check in $\mathcal{G}_k$ with parameter ρ_k

---

Each node $i \in \mathcal{V}_k$ should performs these steps:

1. On each outgoing link $e = (i, j) \in \mathcal{E}_k$ whose capacity is z_e, node i transmits z_e linear combinations of the ρ_k symbols in vector $\mathbf{X_i}$, with the weights for the linear combinations being chosen from $GF(2^{L/\rho_k})$.

   More formally, for *each* outgoing edge $e = (i, j) \in \mathcal{E}_k$ of capacity z_e, a $\rho_k \times z_e$ matrix $\mathbf{C_e}$ is specified as a part of the algorithm. Entries in $\mathbf{C_e}$ are chosen from $GF(2^{L/\rho_k})$. Node i sends to node j a vector $\mathbf{Y_e}$ of z_e symbols obtained as the matrix product $\mathbf{Y_e} = \mathbf{X_i C_e}$. Each element of $\mathbf{Y_e}$ is said to be a "coded symbol". The choice of the matrix $\mathbf{C_e}$ affects the correctness of the algorithm, as elaborated later.

2. On each incoming edge $d = (j, i) \in \mathcal{E}_k$, node i receives a vector $\mathbf{Y_d}$ containing z_d symbols from $GF(2^{L/\rho_k})$. Node i then checks, for each incoming edge d, whether $\mathbf{Y_d} = \mathbf{X_i C_d}$. The check is said to fail iff $\mathbf{Y_d} \neq \mathbf{X_i C_d}$.

3. If checks of symbols received on any incoming edge fails in the previous

step, then node i sets a 1-bit *flag* equal to MISMATCH; else the *flag* is set to NULL. This flag is broadcast in *Step 2.2* above.

In the *Equality Check* algorithm, z_e symbols of size L/ρ_k bits are transmitted on each link e of capacity z_e. Therefore, the *Equality Check* algorithm requires time duration

$$L \ / \ \rho_k. \tag{3.3}$$

Salient Feature of Equality Check Algorithm

In the *Equality Check* algorithm, a single round of communication occurs between adjacent nodes. No node is required to forward packets received from other nodes during the *Equality Check* algorithm. This implies that, while a faulty node may send incorrect packets to its neighbors, it cannot tamper with information sent between fault-free nodes. This feature of *Equality Check* is important in being able to prove its correctness despite the presence of faulty nodes in $\mathcal{G}_k$.

Choice of Parameter ρ_k

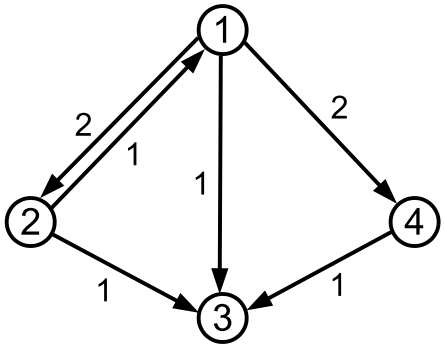
We define a set Ω_k as follows using the disputes identified through the first $(k-1)$ instances of NAB.

$$\begin{aligned}\Omega_k \ = \ \{ \ H \ \mid \ & H \text{ is a subgraph of } \mathcal{G}_k \text{ containing } (n-f) \text{ nodes such that} \\ & \text{no two nodes in } H \text{ have been found } \textit{in dispute} \text{ through} \\ & \text{the first } (k-1) \text{ instances of NAB } \}\end{aligned}$$

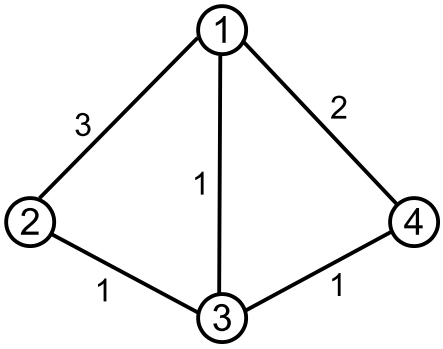
As noted in the discussion of Phase 3 (Dispute Control), fault-free nodes are never found in dispute *with each other* (fault-free nodes may be found in dispute with faulty nodes, however). This implies that $\mathcal{G}_k$ includes all the fault-free nodes, since a fault-free node will never be found in dispute with $f+1$ other nodes. There are at least $n-f$ fault-free nodes in the network. This implies that set Ω_k is non-empty.

Corresponding to a directed graph $H(V,E)$, let us define an undirected graph $\overline{H}(V,\overline{E})$ as follows: (i) both H and $\overline{H}$ contain the same set of vertices,

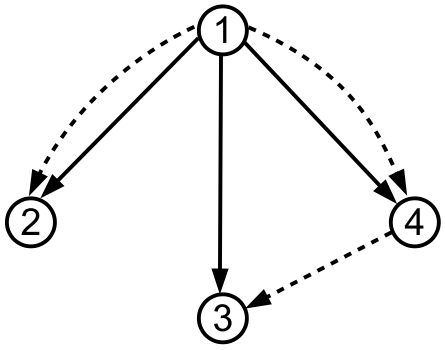
(ii) undirected edge $(i, j) \in \overline{E}$ if either $(i, j) \in E$ or $(j, i) \in E$, and (iii) capacity of undirected edge $(i, j) \in \overline{E}$ is defined to be equal to the sum of the capacities of directed links (i, j) and (j, i) in E (if a directed link does not exist in E, here we treat its capacity as 0). For example, Figure 3.2(b) shows the undirected graph corresponding to the directed graph in Figure 3.2(a).



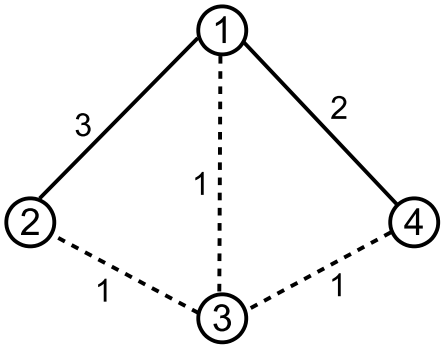
(a) Directed graph G



(b) Undirected graph $\overline{G}$



(c) Two unit-capacity spanning trees in the directed graph. Every directed edge has capacity 1



(d) A spanning tree in the undirected graph shown in dotted edges

Figure 3.2: Different graph representations of a network. Numbers next to the edges indicate link capacities.

Define a set of *undirected* graphs $\overline{\Omega}_k$ as follows. $\overline{\Omega}_k$ contains undirected version of each directed graph in Ω.

$$\overline{\Omega}_k \ = \ \{ \ \overline{H} \ \mid \ H \in \Omega_k \ \}$$

Define

$$U_k = \min_{\overline{H} \in \overline{\Omega}_k} \min_{i,j \in \overline{H}} MINCUT(\overline{H}, i, j)$$

as the minimum value of the MINCUTs between all pairs of nodes in all the undirected graphs in the set $\overline{\Omega}_k$. For instance, suppose that $n = 4$, $f = 1$ and the graph shown in Figure 3.1(a) on page 62 is $\mathcal{G}$, whereas $\mathcal{G}_k$ is the graph shown in Figure 3.1(b). Thus, nodes 2 and 3 have been found in dispute previously. Then, Ω_k and $\overline{\Omega}_k$ each contain two subgraphs, one subgraph corresponding to the node set $\{1, 3, 4\}$, and the other subgraph corresponding to the node set $\{2, 3, 4\}$. In this example, $U_k = 2$.

Parameter ρ_k is chosen such that

$$\rho_k \ \leq \ \frac{U_k}{2}.$$

Under the above constraint on ρ_k, as per (3.3), execution time of *Equality Check* is minimized when $\rho_k = \frac{U_k}{2}$. Under the above constraint on ρ_k, we will prove the correctness of the *Equality Check* algorithm, with its execution time being L/ρ_k.

3.4.1 Correctness of the Equality Check Algorithm

The correctness of Algorithm 3.4.5 depends on the choices of the parameter ρ_k and the set of coding matrices $\{\mathbf{C_e} | e \in \mathcal{E}_k\}$. Let us say that a set of coding matrices is *correct* if the resulting *Equality Check* Algorithm 3.4.5 satisfies the following requirement:

- (EC) **if** there exists a pair of fault-free nodes $i, j \in \mathcal{G}_k$ such that $\mathbf{X_i} \neq \mathbf{X_j}$ (i.e., $x_i \neq x_j$),
  then the 1-bit flag at **at least one** fault-free node is set to MISMATCH.

Recall that $\mathbf{X_i}$ is a vector representation of the L-bit value x_i received by node i in Phase 1 of NAB. Two consequences of the above correctness condition are:

- If some node (possibly source node) misbehaves during Phase 1 leading to outcomes (ii) or (iv) for Phase 1, then at least one fault-free node will set its flag to MISMATCH. In this case, the fault-free nodes (possibly

including the sender) have not received identical L-bit values in Phase 1.

- If no misbehavior occurs in Phase 1 (thus the values received by fault-free nodes in Phase 1 are correct), but MISMATCH flag at some fault-free node is set in *Equality Check*, then misbehavior must have occurred in Phase 2.

The following theorem shows that when $\rho_k \leq U_k/2$, and when L is sufficiently large, there exist of coding matrices $\{\mathbf{C_e}|e \in \mathcal{E}_k\}$ that are correct.

Theorem 3.1 *For $\rho_k \leq U_k/2$, when the entries of the coding matrices $\{\mathbf{C_e}|e \in \mathcal{E}_k\}$ in step 1 of Algorithm 3.4.5 are chosen independently and uniformly at random from $GF(2^{L/\rho_k})$, then $\{\mathbf{C_e}|e \in \mathcal{E}_k\}$ is correct with probability $\geq 1 - 2^{-L/\rho_k}\left[\binom{n}{n-f}(n-f-1)\rho_k\right]$. Note that when L is large enough, $1 - 2^{-L/\rho_k}\left[\binom{n}{n-f}(n-f-1)\rho_k\right] > 0$.*

Proof Sketch: The detailed proof is presented in Appendix B.3. Here we provide a sketch of the proof. The goal is to prove that property (EC) above holds with a non-zero probability. That is, regardless of which (up to f) nodes in $\mathcal{G}$ are faulty, when $\mathbf{X_i} \neq \mathbf{X_j}$ for some pair of fault-free nodes i and j in $\mathcal{G}_k$ during the k-th instance, at least one fault-free node (which may be different from nodes i and j) will set its 1-bit flag to MISMATCH. To prove this, we consider every subgraph of $H \in \Omega_k$ (see definition of Ω_k above). By definition of Ω_k, no two nodes in H have been found in dispute through the first $(k-1)$ instances of NAB. Therefore, H represents one *potential* set of $n-f$ fault-free nodes in $\mathcal{G}_k$. Denote $\mathbf{0_{1\times z}}$ as a $1 \times z$ vector of which all symbols equal to 0. For each edge $e = (i, j)$ in H, steps 1-2 of Algorithm 3.4.5 together have the effect of checking whether or not $(\mathbf{X_i} - \mathbf{X_j})\mathbf{C_e} = \mathbf{0_{1\times z_e}}$. Without loss of generality, for the purpose of this proof, *rename* the nodes in H as $1, \cdots, n-f$. Denote a $1 \times \rho_k$ vector in $GF(2^{L/\rho_k})$

$$\mathbf{D_i} = \mathbf{X_i} - \mathbf{X_{n-f}}$$

for $i = 1, \cdots, (n-f-1)$, then

$$(\mathbf{X_i} - \mathbf{X_j})\mathbf{C_e} = \mathbf{0_{1\times z_e}} \Leftrightarrow \begin{cases} (\mathbf{D_i} - \mathbf{D_j})\mathbf{C_e} = \mathbf{0_{1\times z_e}} & , \quad \text{if } \ i, j < n-f; \\ \mathbf{D_i C_e} = \mathbf{0_{1\times z_e}} & , \quad \text{if } \ j = n-f; \\ -\mathbf{D_j C_e} = \mathbf{0_{1\times z_e}} & , \quad \text{if } \ i = n-f. \end{cases} \tag{3.4}$$

Define

$$\mathbf{D_H} = [\mathbf{D_1}, \mathbf{D_2}, \cdots, \mathbf{D_{n-f-1}}].$$

$\mathbf{D_H}$ is a $1 \times \rho_k(n-f-1)$ vector in $GF(2^{L/\rho_k})$. Let m be the sum of the capacities of all the directed edges in H. As elaborated in Appendix B.3, we define $\mathbf{C_H}$ to be a $(n-f-1)\rho_k \times m$ matrix in $GF(2^{L/\rho_k})$ whose entries are obtained using the elements of $\mathbf{C_e}$ for each edge e in H in an appropriate manner. For the suitably defined $\mathbf{C_H}$ matrix, we can show that the comparisons in steps 1-2 of Algorithm 3.4.5 at all the nodes in $H \in \Omega_k$ are equivalent to checking whether or not

$$\mathbf{D_H C_H} \ = \ \mathbf{0_{1\times m}}. \tag{3.5}$$

We can show that for a particular subgraph $H \in \Omega_k$, when $\rho_k \leq U_k/2$, $m \geq (n-f-1)\rho_k$; and when the set of coding matrices $\{\mathbf{C_e} | e \in \mathcal{E}_k\}$ are generated as described in Theorem 3.1, for large enough L, with non-zero probability $\mathbf{C_H}$ contains a $(n-f-1)\rho_k \times (n-f-1)\rho_k$ square submatrix that is invertible. In this case $\mathbf{D_H C_H} = \mathbf{0_{1\times m}}$ if and only if $\mathbf{D_H} = \mathbf{0_{1\times m}}$, i.e., $\mathbf{X_1} = \mathbf{X_2} = \cdots = \mathbf{X_{n-f}}$. In other words, if all nodes in subgraph H are fault-free, and $\mathbf{X_i} \neq \mathbf{X_j}$ for two fault-free nodes i, j, then $\mathbf{D_H C_H} \neq \mathbf{0_{1\times m}}$ and hence the check in step 2 of Algorithm 3.4.5 fails at some fault-free node in H.

We can then show that, for large enough L, with a non-zero probability, this is also **simultaneously** true for all subgraphs $H \in \Omega_k$. This implies that, for large enough L, correct coding matrices ($\mathbf{C_e}$ for each $e \in \mathcal{E}_k$) can be found. These coding matrices are specified as a part of the algorithm specification. Further details of the proof are in Appendix B.3. □

3.5 Correctness of NAB

For Phase 1 (Unreliable Broadcast) and Phase 3 (Dispute Control), the proof that the outcomes claimed in Section 3.3 indeed occur follows directly from the prior literature cited in Section 3.3 (and elaborated in Appendices B.1 and B.2). Now consider two cases:

- The values received by the fault-free nodes in Phase 1 are *not identical*: Then the correctness of *Equality Check* ensures that a fault-free node will detect the mismatch, and consequently Phase 3 will be performed. As a byproduct of *Dispute Control* in Phase 3, the fault-free nodes will correctly agree on a value that satisfies the *validity* and *agreement* conditions.

- The values received by the fault-free nodes in Phase 1 are identical: If no node announces a mismatch in step 2.2, then the fault-free nodes will agree on the value received in Phase 1. It is easy to see that this is a correct outcome. On the other hand, if some (faulty) node announces a mismatch in step 2.2, then *Dispute Control* will be performed, which will result in correct outcome for the broadcast of the k-th instance.

Thus, in all cases, NAB will lead to correct outcome in each instance.

3.6 Throughput of NAB and Capacity of BB

3.6.1 A Lower Bound on Throughput of NAB for Large Q and L

In this section, we provide the intuition behind the derivation of the lower bound. More detail is presented in Appendix B.4. We prove the lower bound when the number of instances Q and input size L for each instance are both "large" (in an order sense) compared to n. Two consequences of L and Q being large:

- As a consequence of Q being large, the average overhead of *Dispute control* per instance of NAB becomes negligible. Recall that *Dispute Control* needs to be performed at most $f(f+1)$ times over Q executions of NAB.

- As a consequence of L being large, the overhead of 1-bit broadcasts performed in step 2.2 of Phase 2 becomes negligible when amortized over the L bits being broadcast by the source in each instance of NAB.

It then suffices to consider only the time it takes to complete the *Unreliable broadcast* in Phase 1 and *Equality Check* in Phase 2. For the k-th instance of NAB, as discussed previously, the unreliable broadcast in Phase 1 can be done in L/γ_k time units (see definition of γ_k in section 3.3.). We now define

$$\begin{aligned}\Gamma \;=\; \{\; J \;\mid\; & J \text{ is a subgraph of } \mathcal{G} \text{ containing source node 1,} \\ & \text{and } \mathcal{G}_k \text{ may equal } J \text{ in some execution of NAB for some } k \;\}\end{aligned}$$

Appendix B.5 provides a systematic construction of the set Γ. Define the minimum value of all possible γ_k:

$$\gamma^* = \min_{\mathcal{G}_k \in \Gamma} \gamma_k = \min_{\mathcal{G}_k \in \Gamma} \min_{j \in \mathcal{V}_k} MINCUT(\mathcal{G}_k, 1, j).$$

Then an upper bound of the execution time of Phase 1 in all instances of NAB is L/γ^*.

With parameter $\rho_k = U_k/2$, the execution time of the Equality Check in Phase 2 is L/ρ_k. Recall that U_k is defined as the minimum value of the MINCUTs between all pairs of nodes in all undirected graphs in the set $\overline{\Omega}_k$. As discussed in Appendix B.3.2, according to the way $\mathcal{G}_k$ is constructed and the definition of Ω_k, $\mathcal{G}_k$ is a subgraph of $\mathcal{G}_1 = \mathcal{G}$, and $\Omega_k \subseteq \Omega_1$. Then according to the definition of U_k,

$$\begin{aligned} U_k &= \min_{\overline{H} \in \overline{\Omega}_k} \min_{i,j \in \overline{H}} MINCUT(\overline{H}, i, j) \\ &\geq \min_{\overline{H} \in \overline{\Omega}_1} \min_{i,j \in \overline{H}} MINCUT(\overline{H}, i, j) \\ &= U_1 \end{aligned}$$

for all possible $\mathcal{G}_k$. Define

$$\rho^* = \frac{U_1}{2}.$$

Then $\rho_k \geq \rho^*$ for all possible $\mathcal{G}_k$ and the execution time of the Equality Check is upper-bounded by L/ρ^*. So the throughput of NAB for large Q and L can

be lower-bounded by[2]

$$\lim_{L\to\infty} T(\mathcal{G}, L, NAB) \geq \frac{L}{L/\gamma^* + L/\rho^*} = \frac{\gamma^*\rho^*}{\gamma^* + \rho^*}. \tag{3.6}$$

3.6.2 An Upper Bound on Capacity of BB

Theorem 3.2 *In any point-to-point network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the capacity of Byzantine broadcast (C_{BB}) with node 1 as the source satisfies the following upper bound:*

$$C_{BB}(\mathcal{G}) \leq \min(\gamma^*, 2\rho^*).$$

Appendix B.6 presents a proof of this upper bound. Given the throughput lower bound in (3.6) and the upper bound on $C_{BB}(\mathcal{G})$ from Theorem 3.2, we show that the NAB algorithm always achieves throughput at least as high as 1/3 of the capacity, as stated as the following theorem:

Theorem 3.3 *For graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$:*

$$\lim_{L\to\infty} T(\mathcal{G}, L, NAB) \geq \frac{\min(\gamma^*, 2\rho^*)}{3} \geq \frac{C_{BB}(\mathcal{G})}{3}.$$

Moreover, when $\gamma^* \leq \rho^*$:

$$\lim_{L\to\infty} T(\mathcal{G}, L, NAB) \geq \frac{\min(\gamma^*, 2\rho^*)}{2} \geq \frac{C_{BB}(\mathcal{G})}{2}.$$

Proof: Let us denote

$$T_{NAB}(\mathcal{G}) = \frac{\gamma^*\rho^*}{\gamma^* + \rho^*}.$$

From Equality 3.6, we know that $\lim_{L\to\infty} T(\mathcal{G}, L, NAB) \geq T_{NAB}(\mathcal{G})$. We will compare $T_{NAB}(\mathcal{G})$ with $\min(\gamma^*, 2\rho^*)$ – the upper bound on $C_{BB}(\mathcal{G})$ from Theorem 3.2. There are 3 possible cases in terms of the relative values of γ^* and ρ^*:

1. $\gamma^* \leq \rho^*$: Observe that $T_{NAB}(\mathcal{G})$ is an increasing function of both γ^* and ρ^*. For a given γ^*, it is minimized when ρ^* is minimized. So

$$T_{NAB}(\mathcal{G}) \geq \frac{\gamma^{*2}}{\gamma^* + \gamma^*} = \frac{\gamma^*}{2} \geq \frac{C_{BB}(\mathcal{G})}{2}. \tag{3.7}$$

[2]To simplify the analysis above, we ignored propagation delays. Appendix B.4 describes how to achieve this bound even when propagation delays are considered.

The last inequality is due to $\gamma^* \geq C_{BB}(\mathcal{G})$.

2. $\gamma^* \leq 2\rho^*$:

$$T_{NAB}(\mathcal{G}) \geq \frac{\gamma^*\rho^*}{2\rho^* + \rho^*} = \frac{\gamma^*}{3} \geq \frac{C_{BB}(\mathcal{G})}{3}. \tag{3.8}$$

The last inequality is due to $\gamma^* \geq C_{BB}(\mathcal{G})$.

3. $\gamma^* > 2\rho^*$: Since $T_{NAB}(\mathcal{G})$ is an increasing function of γ^*, for a given ρ^*, it is minimized when γ^* is minimized. So

$$T_{NAB}(\mathcal{G}) \quad \geq \quad \frac{2{\rho^*}^2}{2\rho^* + \rho^*} = \frac{2\rho^*}{3} \geq \frac{C_{BB}(\mathcal{G})}{3}. \tag{3.9}$$

The second inequality is due to $2\rho^* \geq C_{BB}(\mathcal{G})$.

□

3.7 The CAB Algorithms

In the previous sections of this chapter, we have shown that the NAB algorithm can achieve Byzantine broadcast throughput arbitrarily close to 1/3 of the upper bound on the capacity of Byzantine broadcast from Theorem 3.2, in a *general* point-to-point network $\mathcal{G}$. In this section, we will show that the upper bound from Theorem 3.1 is tight in the two particular families of networks below by providing algorithms that achieve throughput arbitrarily close to the upper bound of capacity.

- Four-node networks: fully connected network of four nodes and at most one faulty node, with arbitrary distribution of link capacities.
- Symmetric networks: fully connected network in which all link capacities are identical, with arbitrary number of nodes $n \geq 4$ and number of faulty nodes $f < n/3$.

3.7.1 CAB Algorithm for Four-Node Networks

In this section, we use the same notations (such as $\mathcal{G}_k, \mathcal{V}_k, \mathcal{E}_k$) as in previous sections. For four-node networks, Theorem 3.2 reduces to the following:

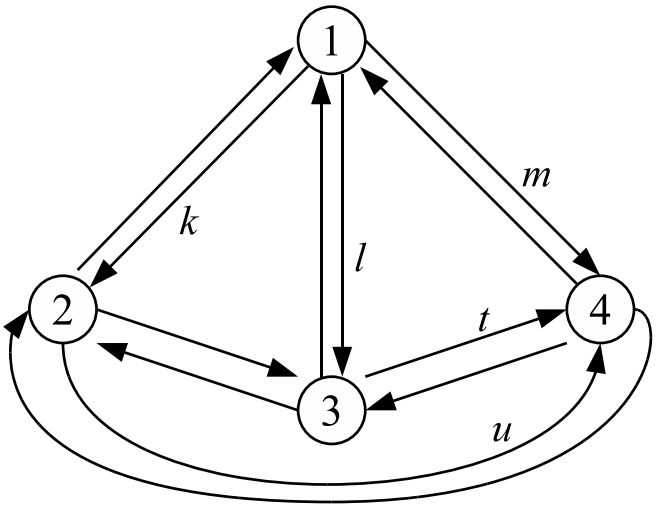


Figure 3.3: Four-node network. Labels denote some link capacities.

Corollary 3.1 *In any four-node network with arbitrary distribution of link capacities, the capacity of Byzantine broadcast (C_{BB4}) is upper-bounded by*

- *the sum capacity of any two incoming links to a peer node i, for $i = 2, 3, 4$.*
- *the sum capacity of any two outgoing links of the source node 1.*

Figure 3.3 shows a complete four-node network. The labels near the various links denote the link capacities. With this notation, the first condition in Corollary 3.1 implies, for instance, that $t + u \geq C_{BB4}$; and the second condition implies, for instance, that $l + m \geq C_{BB4}$.

The CAB algorithm for four-node networks is in fact similar to the NAB algorithm for general point-to-point networks. The main difference is that, by exploring the structure of the four-node networks, the CAB algorithm performs the unreliable broadcast in Phase 1 of NAB and failure detection in Phase 2 of NAB *simultaneously*. In other words, the transmissions for delivering x from source node 1 to the three peers are also used for equality checking. By doing this, the link capacities are fully utilized and throughput matching the upper bound can be achieved.

As in NAB, we represent the L-bit input value x of the k-th instance of Byzantine broadcast as a vector $\mathbf{X}$ of R symbols from Galois Field $GF(2^{L/R})$. The operation of the CAB algorithm is described in pseudo-code in Algorithm

3.7.6. For the following discussion, denote $z_{i,j}$ as the capacity of edge $e = (i, j)$.

Algorithm 3.7.6 CAB Algorithm for four-node networks, with $\mathcal{G}_k$ and parameter R (instance k)

1.Source node 1:

On each outgoing link $e = (1, i) \in \mathcal{E}_k$ whose capacity is z_e, node 1 transmits a vector of z_e coded symbols $\mathbf{Y_e} = \mathbf{XC_e}$, with the weights for the linear combinations being chosen from $GF(2^{L/R})$.

2.Each peer node i that is not in dispute with the source, i.e., $(1, i)$ and $(i, 1)$ are both in $\mathcal{E}_k$:

On each outgoing link $d = (i, j) \in \mathcal{E}_k$ $(j \neq 1)$ whose capacity is z_d, node i *forwards* to node j the *first* $\min\{z_d, z_e\}$ coded symbols of $\mathbf{Y_e}$ received from node 1 in the previous step on link $e = (1, i) \in \mathcal{E}_k$, following an increasing order of the indices.

3.Peer node j that is in dispute with the source (if there is any), i.e., neither $(1, j)$ nor $(j, 1)$ is in $\mathcal{E}_k$:

Denote the other two peers as l and p. Note that links $(j, l), (j, p), (l, j)$ and (p, j) are all in $\mathcal{E}_k$; otherwise, node j must have been identified as faulty and removed. Node l computes $\max\{z_{l,j} - z_{1,l}, 0\}$ coded symbols as linear combinations of the coded symbols it has received from node 1 and node p in steps 1-2, and sends them to node j. Similarly, node p computes and sends $\max\{z_{p,j} - z_{1,p}, 0\}$ symbols to node j. Now node j has received $z_{l,j}+z_{p,j}$ coded symbols from node l and node p in steps 2-3. Then it computes $\min\{z_{j,l}, z_{j,p}\}$ coded symbols as linear combinations of the coded symbols it has received. Then node j sends these set of $\min\{z_{j,l}, z_{j,p}\}$ coded symbols on *both* links (j, l) and (j, p).

4.Each peer node i checks if there exists a unique $1 \times R$ vector $\mathbf{Y_i}$ from $GF(2^{L/R})$ that satisfies all the linear combinations represented by all the coded symbols it has received through steps 1-3. The check is said to fail iff such an unique vector cannot be found.

5.The rest is the same as in the NAB algorithm.

Steps 1-3 here correspond to the unreliable broadcast in Phase 1 and steps 1 of the equality checking algorithm in Phase 2 of the NAB algorithm, and step 4 here corresponds to step 2 of the equality checking algorithm. Similar to the proof of Theorem 3.1, we show that when parameter R is no greater than the upper bound stated in Corollary 3.1 and the value L is sufficiently large, there exist sets of weights for computing the linear combinations such that the resulting Algorithm 3.7.6 is *correct*, where the correctness of the algorithm is defined as in Section 3.4.1. The proof is included in Appendix B.7 for completeness.

Algorithm 3.7.6 can achieve throughput arbitrarily close to the upper bound stated in Corollary 3.1, as per a similar throughput analysis as in Section 3.6 for NAB. Hence,

Theorem 3.4 *In any four-node network, the capacity of Byzantine broadcast is equal to the minimum value of*

- *the sum capacity of any two incoming links to a peer node i, for $i = 2, 3, 4$.*
- *the sum capacity of any two outgoing links of the source node 1.*

Additionally, Algorithm 3.7.6 can achieve throughput arbitrarily close to the capacity.

3.7.2 CAB Algorithm for Symmetric Networks

In a symmetric network of n nodes, every pair of nodes i and j in $\mathcal{V}$ is connected with a pair of directed links $(i, j), (j, i) \in \mathcal{E}$. Also, every edge $e \in \mathcal{E}$ has the same capacity, i.e., $z_e = z$ for some positive constant z. In such a symmetric network with up to $f < n/3$ faulty nodes, Theorem 3.2 reduces to the following:

Corollary 3.2 *In any symmetric network with up to $f < n/3$ faulty nodes, the capacity of Byzantine broadcast (C_{SYM}) is upper-bounded by*

$$C_{SYM} \leq (n - f - 1)z,$$

where z is the capacity of each link in the network.

The CAB algorithm for symmetric networks is in fact almost identical to the CBB Algorithm from Section 2.7.2, with the following modifications:

- Replace the $(2(n-1), n-f)$ distance-$(n+f-1)$ Reed-Solomon code C_{n-f} used in CBB with a $(n-1, n-f-1)$ distance-$(f+1)$ Reed-Solomon code, denoted as C_{n-f-1}. Also accordingly replace $(n-f)$ in CBB with $(n-f-1)$.

- In `Line 5` of CBB, instead of receiving $R_j[j+(n-1)]$ from each peer j trusted by both node i and source node n, we require every node j that is trusted by both node i and the source node n to compute a linear combination of the coded symbols it has received through `Lines 1-4` and send it to node i.

- Dispute control is done in the same way as the NAB algorithm does.

With these modifications, the algorithm achieves throughput arbitrarily close to $(n-f-1)z$, as per similar proof of correctness for CBB and throughput analysis as in Section 3.6 for NAB. Hence,

Theorem 3.5 *In any symmetric network with up to $f < n/3$ faulty nodes and link capacity z, the capacity of Byzantine broadcast is equal to $(n-f-1)z$.*

Additionally, the CBB algorithm with the above modifications can achieve throughput arbitrarily close to the capacity.

3.8 Network-Aware Byzantine Consensus

In previous sections of this chapter, we have discussed the design of network-aware Byzantine broadcast algorithms in point-to-point networks. For the Byzantine consensus problem, we can define throughput and capacity in point-to-point networks in a similar way as for the Byzantine broadcast problem. Also, the same 3-phase structure used in NAB can be used to develop network-aware Byzantine consensus algorithms in general point-to-point networks.

The main difference between Byzantine broadcast and Byzantine consensus is that in the consensus problem there is no designated node acting as the source of information as in the broadcast problem, and every node's input

value will possibly contribute to the final output value. So in Phase 1 for the consensus problem, instead of performing unreliable broadcast from the source node, we need to first identify a subset of nodes that appear to have identical input values, and then try to agree with that value. For this, we need to perform steps similar to the Matching stage of the CBC algorithm, with appropriate modifications so that link capacities are better utilized. After that, we can perform failure detection in Phase 2 and dispute control in Phase 3 (if failure is detected) in the same way as in NAB.

We have developed an upper bound on the capacity of Byzantine consensus in general point-to-point networks. For four-node networks, we have developed a capacity-achieving consensus algorithm. Details of the upper bound and algorithm can be found in our paper [11]

3.9 Summary

In this chapter, we study the design of network-aware Byzantine agreement algorithms. We derive an upper bound on the capacity of Byzantine broadcast for general point-to-point networks. A network-aware Byzantine broadcast algorithm NAB is proposed and proved to achieve throughput at least 1/3 of the capacity in general point-to-point networks. In two particular families of point-to-point networks, capacity-achieving Byzantine broadcast algorithms are developed.

CHAPTER 4

MULTIPARTY EQUALITY FUNCTION COMPUTATION

4.1 Introduction

In this chapter, we study the problem of computing the following multiparty equality function (MEQ):

$$MEQ(x_1, \cdots, x_n) = \begin{cases} 0 & \text{if } x_1 = \cdots = x_n \\ 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

The input vector $x = (x_1, \cdots, x_n)$ is distributed among $n \geq 2$ nodes, with only x_i known to node i, and each x_i chosen from the set $\{1, \cdots, K\}$, for some integer $K \geq 1$.

Communication Complexity: The notion of communication complexity (CC) was introduced by Yao in 1979 [17]. They investigated the problem of quantifying the number of bits that two separated parties need to communicate between themselves in order to compute a function whose inputs, namely X and Y, are distributed between them.

The communication cost of a protocol P, denoted as $C(P)$, is the number of bits exchanged for the worst case input pair. The communication complexity of a Boolean function $f : X \times Y \mapsto \{0, 1\}$ is the minimum of the cost of the protocols for f.

Multiparty Function Computation: The notion of communication complexity can be easily generalized to a multiparty setting, i.e., when the number of parties > 2.

The communication complexity of a Boolean function $f : X_1 \times \cdots \times X_n \mapsto \{0, 1\}$, is the minimum of the cost of the protocols for f.

There is more than one communication model for the multiparty problems.

Two commonly used models are the "number on the forehead" model [53] and the "number in hand" model. Consider function $f : X_1 \times \cdots \times X_n \mapsto \{0, 1\}$ and input $(x_1, x_2, \cdots, x_n)$ where each $x_i \in X_i$. In the number on the forehead model, the i-th party can see all the x_j such that $j \neq i$; while in the number in hand model, the i-th party can only see x_i. As in the two-party case, the n parties have an agreed-upon protocol for communication, and all this communication is posted on a "public blackboard". In these two models, the communication may be considered as being *broadcast* using the public blackboard, i.e., when any party sends a message, all other parties receive the same message. Tight bounds often follow from considering two-way partitions of the set of parties.

In this chapter, we consider a different point-to-point communication model, in which nodes communicate over *private* point-to-point links. This means that when a party transmits a message on a point-to-point link, only the party on the other end of the link receives the message. This model makes the problem significantly different from that with the broadcast communication model. We are interested in the communication complexity of the MEQ problem under the point-to-point communication model.

4.2 Related Work

The 2-party version of the MEQ problem (i.e., $n = 2$), which is usually referred to as the EQ problem, was first introduced by Yao in [17]. It is shown that the communication complexity of the EQ problem with *deterministic* algorithms is $\log K$ [54]. The complexity of the EQ problem can be reduced to $O(\log \log K)$ if randomized algorithms are allowed [54]. MEQ problem with $n \geq 3$ has been studied under the *number on the forehead* model and the *number in hand* model, both assuming a "public blackboard" for broadcast communications. The MEQ problem with $n \geq 3$ can be solved with cost of 2 bits [54] under the number on the forehead head model, while it requires $\Theta(\log K)$ bits under the number in hand model. On the other hand, the result changes significantly if we consider the point-to-point communication model used in this chapter (it is easy to show at least $\Omega(n \log K)$ bits are needed).

The MEQ problem is related to the Set Disjointness problem and the con-

sensus problem [55]. In the n-party Set Disjointness problem, we have n parties, and given subsets $S_1, \ldots, S_n \subseteq \{1, \ldots, K\}$, and the parties wish to determine if $S_1 \cap \cdots \cap S_n = \phi$ without communicating many bits. The disjointness problem is closely related to our MEQ problem. Consider the two-party set disjointness problem with subsets S_1 and S_2. Let x_1 and x_2 be the binary representations of S_1 and S_2, respectively. Then it is not hard to show that $x_1 = x_2$ is equivalent to $S_1 \cap \overline{S_2} = \phi$ and $\overline{S_1} \cap S_2 = \phi$. The multi-party set disjointness problem has been widely studied under the "number on the forehead" and broadcast communication model, e.g. [56, 57]. The set disjointness problem has also been studied under the "number in hand" model and point-to-point communication model (i.e., the same models we are using in this chapter), with randomized algorithms. In [58], a lower bound of $\Omega(K/n^4)$ on its communication complexity is proved for randomized algorithms. The lower bound was then improved to $\Omega(K/n^2)$ in [59]. In [60], the authors established a further improved near-optimal lower bound of $\Omega(K/(n \log n))$. Nevertheless, these papers focus on the order of the communication complexity of randomized algorithms. On the other hand, in this chapter, our goal is to characterize the *exact* communication complexity of *deterministic* algorithms.

In the Byzantine consensus problem, n parties, each of which is given an input x_i of $\log K$ bits, want to agree on a common output value x of $\log K$ bits under the point-to-point communication model, despite the fact that up to t of the parties may be *faulty* and deviate from the algorithm in arbitrary fashion [55]. The core of the consensus problem is to make sure that all fault-free parties' outputs are identical, which is essentially what the MEQ problem tries to solve. In our recent report [7], we established a lower bound on the communication complexity of the Byzantine consensus problem of n parties as a function of the communication complexity of the MEQ problem of $n - f$ parties. This motivates the MEQ problem under the point-to-point communication model. The consensus problem has also been studied under a slightly different fault-free model [61]. Authors of [61] investigated the fault-free consensus problem, which is essentially solving the MEQ problem with 1-bit inputs, i.e., $K = 2$, in tree topologies. We consider the problem under a more general setting with arbitrary K and do not assume any structure of the communication topology.

4.3 Models and Problem Definition

4.3.1 Communication Model

In this chapter, we consider a point-to-point communication model. We assume a synchronous fully connected network of n nodes. We assume that all point-to-point communication channels/links are *private* such that when a node transmits, only the designated recipient can receive the message. The identity of the sender is known to the recipient.

4.3.2 Protocol

A protocol P is a schedule that consists of a sequence of steps. In each step l, as specified by the protocol, a pair of nodes are selected as the *transmitter* and *receiver*, denoted respectively as T_l and R_l. The transmitter T_l will send a message the receiver R_l. The message being sent is computed as a function $m_l(x_{T_l}, T_l^+(l))$, where x_{T_l} denotes T_l's input, and $T_l^+(l)$ denotes all the messages T_l has received so far. When it is clear from the context, we will use T_l^+ to denote $T_l^+(l)$ to simplify the presentation.

In this chapter, we design protocols that are ***static*** : the triple $\langle T_l, R_l, m_l(\cdot)\rangle$ are pre-determined by the protocol and are independent of the inputs. In other words, in step l, no matter what the inputs are, the transmitter, receiver, and the function according to which the transmitter compute the message are the same. Since the schedule is fixed, a static protocol can be represented as a sequence of $L(P)$ steps: $\{\alpha_1, \alpha_2, \cdots, \alpha_{L(P)}\}$, where $\alpha_l = \langle T_l, R_l, m_l(x_{T_l}, T_l^+)\rangle$ in the l-th step. $L(P)$ is called the length of the protocol P, and P always terminates after the $L(P)$-th step. Denote $S_l(P)$ as the cardinality of $m_l()$, i.e., the number of possible channel symbols needed in step l of a static protocol P, considering all possible inputs. Then the communication cost of a static protocol P is determined by

$$C(P) = \sum_{l=1}^{L(P)} \log_2 S_l(P). \tag{4.2}$$

If only binary symbols are allowed, $S_l(P) = 2$ for all l, and $C(P)$ becomes $L(P)$.

4.3.3 Problem Definitions

We define two versions of the MEQ problem.

MEQ-AD (Anyone Detects): We consider protocols in which every node i decides on a one-bit output $EQ_i \in \{0, 1\}$. A node i is said to have detected a *mismatch* (or inequality of inputs) if it sets $EQ_i = 1$. A protocol P is said to solve the MEQ-AD problem if and only if **at least one** node detects a mismatch when the inputs to the n nodes are not identical. More formally, the following property must be satisfied when P terminates:

$$EQ_1 = \cdots = EQ_n = 0 \Leftrightarrow MEQ(x_1, \cdots, x_n) = 0. \tag{4.3}$$

MEQ-CD (Centralized Detect): The second class of protocols we consider are the ones in which one particular node is assigned to decide on an output. Without loss of generality, we can assume that node n has to compute the output. Then a protocol P is said to solve the MEQ-CD problem if and only if, when P terminates, node n computes output EQ_n such that

$$EQ_n = MEQ(x_1, \cdots, x_n). \tag{4.4}$$

Communication Complexity: Denote $\Gamma_{AD}(n, K)$ and $\Gamma_{CD}(n, K)$ as the sets of all protocols that solve the MEQ-AD and MEQ-CD problems with n nodes, respectively. We are interested in finding the communication complexity of the two versions of the MEQ problem, which is defined as the infimum of the communication cost of protocols in $\Gamma_{AD}(n, K)$ and $\Gamma_{AD}(n, K)$, i.e.,

$$C_{AD}(n, K) = \inf_{P \in \Gamma_{AD}(n,K)} C(P), \text{ and } C_{CD}(n, K) = \inf_{P \in \Gamma_{CD}(n,K)} C(P).$$

Communication Complexity with General Protocols: In general, a protocol that solves the MEQ problem may not necessarily be static. The schedule of transmissions might be determined dynamically on-the-fly, depending on the inputs. So the transmitter and receiver in a particular step l can be different with different inputs. Since the set of all static protocols is a subset of all general protocols, the communication complexities of the two versions of the MEQ problem are bounded from above by the cost of

static protocols. The purpose of this chapter is to show that there exist instances of the MEQ problem whose communication complexity is lower than the intuitive upper bound we are going to present in the next section. For this purpose, it suffices to show that, even if we constrain ourselves to static protocols, some MEQ problems can still be solved with cost lower than the upper bound. In sections 4.6 to 4.7, such examples of static protocols are presented.

4.4 Upper Bound of the Complexity

An upper bound of the communication complexity of both versions of the MEQ problem is $(n-1)\log_2 K$, for all positive integer $n \geq 2$ and $K \geq 1$. This can be proved by a trivial construction: in step i, node i sends x_i to node n, for all $i < n$. The decisions are computed according to

$$EQ_i = \begin{cases} MEQ(x_1, \cdots, x_n) & , i = n; \\ 0 & , i < n. \end{cases} \tag{4.5}$$

It is obvious that this protocol solves both the MEQ-AD and MEQ-CD problems with communication cost $(n-1)\log_2 K$, which implies $C_{AD(CD)}(n, K) \leq (n-1)\log_2 K$. In particular, when $K = 2^k$, we have $C_{AD(CD)}(n, 2^k) \leq (n-1)k$.

For the two-party equality problem ($n = 2$), this bound is tight [54], for arbitrary K. The bound is also tight when $K = 2$ (binary inputs). $(n-1)\log_2 2 = n-1$ bits are necessary when $K = 2$, since any protocol with communication cost $< n-1$ will have at least one node not communicating with any other node at all, making it impossible to solve the MEQ problem. However, in the following sections, we are going to show that the $(n-1)\log K$ bound is not always tight, by presenting a static protocol that solves instances of the MEQ problem with communication cost lower than $(n-1)\log_2 K$.

4.4.1 Loose Lower Bound of Complexity using Traditional Techniques

In most of the existing literature on multiparty communication complexity, the "number on the forehand" model or a broadcast communication model is assumed. Under these models, when a node transmits, all other nodes receive the same message. This broadcasting property makes it possible to consider two-way partitions of the set of nodes since the nodes in each partition share the same information being broadcast and can be viewed as one virtual node. Thus, results from two-party communication complexity can be extended to the multiparty case, and tight bounds (rather than just capturing the order) can then be obtained.

However, the above technique no longer works well in obtaining tight bounds under our point-to-point communication model. For example, the complexity of the two-party equality problem of k-bit inputs ($n = 2, K = 2^k$) can be proved to be k with the "fooling set" argument [54]: Suppose in contradiction that there exists a protocol of complexity at most $C(P) < k$ that solves the two-party equality problem. Then there are at most $2^{C(P)} \leq 2^k - 1$ communication patterns possible between the two nodes. Consider all sets of 2^k pairs of input values (x, x). Using the pigeonhole principle we conclude there exist two pairs (x, x) and (x', x') on which the communication patterns are the same. It is easy to see that the communication pattern of (x, x') is also the same as (x, x). Hence, the nodes' final decisions on (x, x) must agree with their decisions on (x, x'). But then the protocol must be incorrect, since $EQ(x, x') = 1 \neq EQ(x, x)$.

The "fooling set" argument above can be extended to the case with $n > 2$ nodes and arbitrary $M \geq 1$: partitioning the n nodes into two sets (say L and R), there must be at least M patterns of communication between the two sets L and R. By applying this argument to all possible two-partitions such that $|L| = 1$ and $|R| = n - 1$, we can obtain a lower bound on the communication complexity as

$$C_{AD(CD)}(n, K) \geq \frac{n}{2} \log_2 K. \tag{4.6}$$

This lower bound is within a factor of $1/2$ of the upper bound we obtain previously, which implies that $C_{AD(CD)}(n, K) = \Theta(n \log_2 K)$. However, we

can show that the lower bound of $\frac{n}{2}\log_2 K$ is generally not achievable. An example for this is the MEQ(3,4) problem. It can be shown that $C_{AD}(3,4) = C_{CD}(3,4) = 4$, while $\frac{n}{2}\log_2 K = 3$. Details can be found in Appendix C.1.

The example above has demonstrated that, under our point-to-point communication model, we can no longer extend results from two-party communication complexity to multiparty version for tight bounds in the way it has been done under the broadcast communication models. The main reason for this is the lack of modeling of the "networking" aspect of the problem in both the two-party model and the broadcast communication models. In the two-party model, since there are only two nodes, no networking is necessary. In the broadcast communication models, all the nodes share a lot information from the broadcast and have roughly the same view of system, which makes it a not-so-distributed network. On the other hand, under our point-to-point communication modes, each node may only receive information from a subset of nodes; it is even possible that two nodes may receive information from two disjoint sets of nodes. As a result, different nodes can have very different views of the system. This makes the problem of finding the tight bound of communication complexity difficult, and new techniques may be required.

4.5 Equivalent MEQ-AD Protocols

In the rest of this chapter, except for Section 4.8, we will focus on static protocols that solve the MEQ-AD problem.

It is not hard to see that a static protocol P can be interpreted as a directed multi-graph $G(V, E(P))$, where the set of vertices $V = \{1, \cdots, n\}$ represents the n nodes, and the set of directed edges $E(P) = \{(T_1, R_1), \cdots, (T_{L(P)}, R_{L(P)})\}$ represents the transmission schedule in each step. From now on, we will use the terms *protocol* and *graph* interchangeably, as well as the terms *transmission* and *link*. Figure 4.1(a) gives an example of the graph representation of a protocol for $n = 4$. In Figure 4.1(a), the numbers next to the directed links indicate the corresponding step numbers.

Two protocols P and P' in are said to be **equivalent** if their costs are equal, i.e., $C(P) = C(P')$. The following lemma says that we can flip the direction of any edge in $E(P)$ and obtain a protocol P' that is equivalent to P.

Lemma 4.1 *Given any static protocol P for MEQ-AD of length $L(P)$, and any positive integer $l \leq L(P)$, there exists an equivalent static protocol P' of the same length, such that $E(P)$ and $E(P')$ are identical, except that in the l-th step, the transmitter and receiver are swapped, i.e.,*

$$\begin{aligned} E(P') &= E(P)\backslash\{(T_l, R_l)\} \cup \{(R_l, T_l)\} \\ &= \{(T_1, R_1), \cdots, (T_{l-1}, R_{l-1}), (R_l, T_l), (T_{l+1}, R_{l+1}), \cdots, (T_{L(P)}, R_{L(P)})\}. \end{aligned}$$

Proof: Given the integer l and a protocol

$$P = \{\alpha_1, \cdots, \alpha_{l-1}, \alpha_l, \alpha_{l+1}, \cdots, \alpha_{L(P)}\}$$

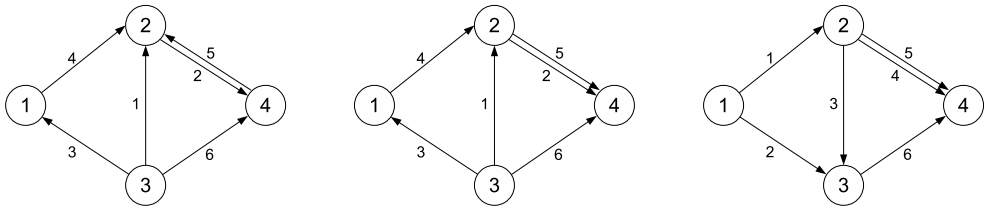
with $\alpha_l = (T_l, R_l, m_l(x_{T_l}, T_l^+))$, we construct

$$P' = \{\alpha'_1, \cdots, \alpha'_{l-1}, \alpha'_l, \alpha'_{l+1}, \cdots, \alpha'_{L(P)}\}$$

by modifying P as follows:

- $\alpha'_j = \alpha_j$ for $1 \leq j \leq l-1$.
- $\alpha'_l = (R_l, T_l, m'_l(x_{R_l}, R_l^+))$. Here $m'_l(x_{R_l}, R_l^+) = m_l(x_{T_l}, T_l^+)|_{x_1=\cdots=x_n=x_{R_l}}$ is the symbol that party R_l *expects* to receive in step l of protocol P, assuming all parties have the same input as x_{R_l}.
- $\alpha'_j = (T_j, R_j, m'_j(x_{T_j}, T_j^+))$ for $j > l$.
  - If $T_j = R_l$, $m'_j(x_{T_j}, T_j^+) = m_j(x_{T_j}, T_j^+)|_{m_l(x_{T_l}, T_l^+)=m'_l(x_{R_l}, R_l^+)}$ is the symbol that party R_l sends in step j, **pretending** that it has received $m'_l(x_{R_l}, R_l^+)$ in step l of P.
  - If $T_j \neq R_l$, $m'_j(x_{T_j}, T_j^+) = m_j(x_{T_j}, T_j^+)$.
- To compute the output, T_l first computes EQ_{T_l} in the same way as in P. Then T_l sets $EQ_{T_l} = 1$ if $m'_l(x_{R_l}, R_l^+) \neq m_l(x_{T_l}, T_l^+)$, else no change. That is, T_l sets EQ_l to 1 if the symbol it receives from R_l in step l of P' differs from the symbol T_l would have sent to R_l in step l of P. The other nodes compute their outputs in the same way as in P.

To show that P and P' are equivalent, consider the two cases:



(a) Graph representation of P

(b) Equivalent protocol of P with Step 5 flipped

(c) iid equivalent protocol of P

Figure 4.1: Example of graph representation of a protocol P and its equivalent protocols. The numbers next to the links indicate the corresponding step number.

- $m'_l(x_{R_l}, R_l^+) = m_l(x_{T_l}, T_l^+)$: It is not hard to see that in this case, the execution of every step is identical in both P and P', except for step l. So for all $i \neq T_l$, EQ_i is identical in both protocols. Since $m'_l(x_{R_l}, R_l^+) = m_l(x_{T_l}, T_l^+)$, EQ_{T_l} remains unchanged, so it is also identical in both protocols.

- $m'_l(x_{R_l}, R_l^+) \neq m_l(x_{T_l}, T_l^+)$: Observe that these two functions can be different only if the n inputs are not all identical. So it is correct to set $EQ_{T_l} = 1$.

□

In Figure 4.1(b), the graph for an equivalent protocol obtained by flipping the link corresponding to the 5-th step of the 4-node example in Figure 4.1(a) is presented.

Let us denote all the symbols a node i receives from and sends to the other nodes throughout the execution of protocol P as i^+ and i^-, respectively. It is obvious that i^- can be written as a function $M_i(x_i, i^+)$, which is the union of $m_l(x_i, i^+(l))$ over all steps l in which node i is the transmitter. If a protocol P satisfies $M_i(x_i, i^+) = M_i(x_i)$ for all i, we say P is **individual-input-determined (iid)**. The following theorem shows that there is always an iid equivalent for every protocol.

Theorem 4.1 *For every static protocol P for MEQ-AD, there always exists an iid equivalent static protocol P^*, which corresponds to an acyclic graph.*

Proof: According to Lemma 4.1, we can flip the direction of any edge in $E(P)$ and obtain a new protocol which is equivalent to P. It is to be noted

that we can keep flipping different edges in the graph, which implies that we can flip any subset of $E(P)$ and obtain a new protocol equivalent to P.

In particular, we consider a protocol equivalent to P, whose corresponding graph is acyclic, and for all $(i,j) \in E(P)$, the property $i < j$ is satisfied. In this protocol, every node i has no incoming links from any node with index greater than i. This implies that the messages transmitted by node i are independent of the inputs to nodes with larger indices. Thus we can re-order the transmissions of this protocol such that node 1 transmits on all of its out-going links first, then node 2 transmits on all of its out-going links, ..., node $n-1$ transmits to n at the end. Name the new protocol Q. Obviously Q is equivalent to P.

Since we can always find a protocol Q equivalent to P as described above, all we need to do now is to find P^*. If Q itself is iid, then $P^* = Q$ and we are done. If not, we obtain P^* in the following way (using function M'), which is similar to how we obtain the equivalent protocol P' in Lemma 4.1:

- For node 1, since it receives nothing from the other nodes, $M_1(x_1, 1^+) = M_1(x_1)$ is trivially true.

- For node $1 < i < n$, we modify Q as follows: node i computes its out-going message as a function $M_i'(x_i) = M_i(x_i, i^+|_{x_1=\cdots=x_n=x_i})$, where $i^+|_{x_1=\cdots=x_n=x_i}$ are incoming messages node i expects to receive, assuming that all parties have the same input x_i. At the end of the protocol, node i checks if $i^+|_{x_1=\cdots=x_n=x_i}$ equals to the actual incoming symbols i^+. If they match, nothing is changed. If they do not match, the inputs cannot be identical, and node i can set $EQ_i = 1$. (The correctness of this step may be easier to see by induction: apply this modification one node at a time, starting from node 1 to node $n-1$.)

□

Theorem 4.1 shows that, to find the least cost of static protocols, it is sufficient to investigate only the static protocols that are iid and the corresponding communication graph is acyclic. From now on, such protocols are called iid static protocols for MEQ-AD. Figure 4.1(c) shows an iid static protocol that is equivalent to the one shown in Figure 4.1(a).

4.6 MEQ-AD(3,6)

Let us first consider MEQ-AD(3,6), i.e., the case where 3 nodes (say A, B and C) are trying to solve the MEQ-AD problem when each node is given input from one out of six values, namely $\{1, 2, 3, 4, 5, 6\}$. According to Theorem 4.1, for any protocol that solves this MEQ-AD problem, there exists an equivalent iid partially ordered protocol in which node A has no incoming link, node B only transmits to node C, and node C has no out-going link. We construct one such protocol that solves MEQ-AD(3,6) and requires only 3 channel symbols, namely $\{1, 2, 3\}$, per link. Denoting the channel symbol being sent over link ij as s_{ij}, the schedule of the proposed protocol is: (1) Node A sends $s_{AB}(x_A)$ to node B; (2) Node A sends $s_{AC}(x_A)$ to node C; and (3) Node B sends $s_{BC}(x_B)$ to node C. Table 4.1 shows how s_{ij} is computed as a function of x_i.

Table 4.1: A protocol for MEQ-AD(3,6)

x	1	2	3	4	5	6
s_{AB}	1	1	2	2	3	3
s_{AC}	1	2	2	3	3	1
s_{BC}	1	2	3	1	2	3

Now consider the outputs. Node A simply sets $EQ_A = 0$. For nodes B and C, they just compare the channel symbol received from each incoming link with the *expected* symbol computed with its own input value, and detect a mismatch if the received and expected symbols are not identical. For example, node B receives $s_{AB}(x_A)$ from node A. Then it detects a mismatch if the $s_{AB}(x_A) \neq s_{AB}(x_B)$.

It can be easily verified that if the three input values are not all identical, at least one of nodes B and C will detect a mismatch. Hence the MEQ-AD(3,6) problem is solved with the proposed protocol. The communication cost of this protocol is

$$3 \log_2 3 = \log_2 27 \approx 0.92 \times 2 \log_2 6. \tag{4.7}$$

Notice that in this case, the upper bound from Section 4.4 equals to $(3-1)\log_2 6 = 2\log_2 6$. So we have found a static MEQ-AD protocol whose communication cost is lower than the upper bound. In fact, this protocol

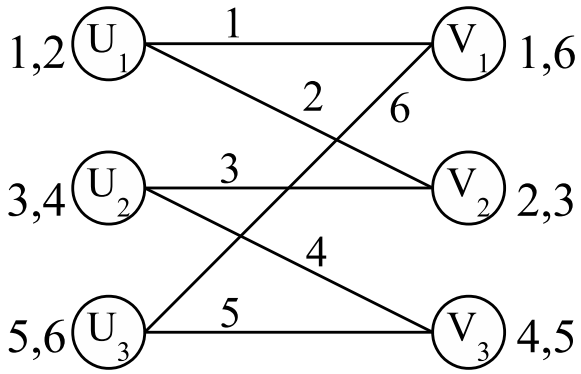


Figure 4.2: Bipartite graph for the MEQ-AD(3,6) protocol in Table 4.1.

is optimal in the sense that it can be shown to achieve the minimum communication cost among all static protocols We prove the optimality of this protocol using an edge coloring argument.

4.6.1 Edge Coloring Representation of MEQ-AD(3,K)

From Sections 4.5, we have shown that it is sufficient to study 3-node systems where messages are transmitted only on links AB, AC and BC. Let us denote $|s_{AB}|$, $|s_{AC}|$ and $|s_{BC}|$ as the number of different symbols being transmitted on links AB, AC and BC, respectively.

Theorem 4.2 *The existence of a MEQ-AD(3,K) static protocol P with cost $C(P)$ is equivalent to the existence of a simple bipartite graph $G(U, V, E)$ and a distance-2 edge coloring scheme W, such that $|U| \times |V| \times |W| = 2^{C(P)}$, given $|E| = K$, $|U| \times |V| \geq K$, $|U| \times |W| \geq K$ and $|V| \times |W| \geq K$. Here U and V are disjoint sets of vertices, E is the set of edges, $|U| = |s_{AB}|$ and $|V| = |s_{AC}|$ are the sizes of sets U and V, and $|W| = |s_{BC}|$ is the number of colors used in W.*

The detailed proof can be found in Appendix C.1. According to Theorem 4.2, we can conclude that the problem of finding a least cost static protocol for $MEQ - AD(3, K)$ is equivalent to the problem of finding the minimum of $|U| \times |V| \times |W|$ for the bipartite graphs and distance-2 coloring schemes that satisfy the above constraints.

Using Theorem 4.2, to show that $C_{AD}(3, 6) = \log_2 27$, we only need to show that for every combination of $|U| \times |V| \times |W| < 27$, there exists no

bipartite graph $G(U, V, E)$ and distance-2 coloring scheme W that satisfy the conditions as described in Theorem 4.2. It is not hard to see that there are only two combinations (up to permutation) that satisfy all conditions and have product less than 27: (2, 3, 3) and (2, 3, 4). Notice that in both cases, $|E| = |U| \times |V|$, where every pair of edges is within distance 2 of each other, which means that the corresponding graph $G(U, V, E)$ can only be distance-2 edge colored with at least $|E| = 6 > 4 > 3$ colors. So neither (2, 3, 3) nor (2, 3, 4) satisfies the aforementioned conditions. Hence, together with the protocol presented before, we can conclude that $C_{AD}(3, 6) = \log_2 27$. The bipartite graph corresponding to Table 4.1 is illustrated in Figure 4.2. Near the nodes U_i (or V_i) we show the set of value x's such that $s_{AB}(x) = i$ (or $s_{AC}(x) = i$). The number near each edges is the input value corresponding to that edge.

4.7 MEQ-AD(3,2^k)

Now we construct a protocol when the number of possible input values $K = 2^k, k \geq 1$ and only binary symbols can be transmitted in each step, using the MEQ-AD(3,6) protocol we just introduced in the previous sections as a building block.

First, we map the 2^k input values into 2^k different vectors in the vector space $\{1, 2, 3, 4, 5, 6\}^h$, where $h = \lceil \log_6 2^k \rceil = \lceil k \log_6 2 \rceil$. Then h instances of the MEQ-AD(3,6) protocol are performed in parallel to compare the h dimensions of the vector. Since 3 channels symbols are required for each instance of the MEQ-AD(3,6) protocol, we need to transmit a vector from $\{1, 2, 3\}^h$ on each of the links AB, AC and BC. One way to do so is to encode the 3^h possible vectors from $\{1, 2, 3\}^h$ into $b = \lceil \log_2 3^h \rceil = \lceil h \log_2 3 \rceil$ bits, and transmit the b bits through the links. Since the h instances of MEQ-AD(3,6) protocols solve the MEQ-AD(3,6) problem for each dimension, altogether they solve the MEQ-AD(3,2^k) problem. The communication cost this protocol can be computed as [19]

$$C(P) = 3\lceil h \log_2 3 \rceil < (0.92 \times 2k) + 7.755. \tag{4.8}$$

From Equation 4.8, we can see that when k is large enough, the commu-

nication cost of this protocol is upper-bounded by 0.92 times of the upper bound $2\log_2 2^k = 2k$ from Section 4.4. The way in which the above protocol is constructed can be generalized to obtain a MEQ-AD(3,K) protocol P with similar cost

$$C(P) < (0.92 \times 2\log_2 K) + \Delta \tag{4.9}$$

for arbitrary value of K, where Δ is some positive constant.

4.8 About MEQ-CD

In this section, we will show that $C_{CD}(n, K)$ *roughly* equals to $C_{AD}(n, K)$:

$$C_{AD}(n, K) \leq C_{CD}(n, K) \leq C_{AD}(n, K) + n - 1. \tag{4.10}$$

We have shown the first inequality in Section 4.3.3. The second inequality can be proved by the following simple construction: Consider any protocol P for MEQ-AD, and construct a protocol P' by having node i send EQ_i to node n by the end of P, for all $i < n$. Node n collects the $n-1$ decisions from all other nodes and computes the final decision

$$EQ'_n = \max\{EQ_1, \cdots, EQ_n\}. \tag{4.11}$$

It is easy to see that $EQ'_n = MEQ(x_1, \cdots, x_n)$. So $P' \in \Gamma_{CD}(n, K)$. Since $C(P') = C(P) + n - 1$, the second inequality is proved. From Equation 4.9 it then follows that for large enough K, the MEQ-CD(3,K) problem can also be solved with communication strictly less than $2\log_2 K$ bits. The performance can be improved somewhat by exploiting communication that may be already taking place between node n and the other nodes. For example, to solve MEQ-CD(3,6), instead of having nodes A and B sending 1 extra bit to node C at the end of the MEQ-AD(3,6) protocol in Section 4.6, we only need to add one possible value to s_{BC}, namely $s_{BC} \in \{1, 2, 3, 4\}$, where $s_{BC} = 4$ means that node B has detected a mismatch. The cost of this protocol is $2\log_2 3 + \log_2 4 = 2\log_2 3 + 2 < 3\log_2 3 + 2$. The same approach can also be applied to the MEQ-AD(3,2^k) protocol from Section 4.7 by making $|s_{BC}| = \lceil \log_2(3^h + 1) \rceil$, and obtain an MEQ-CD(3,2^k) protocol with cost of $2\lceil h \log_2 3 \rceil + \lceil \log_2(3^h + 1) \rceil$ bits, which is almost the same as $3\lceil h \log_2 3 \rceil$ for

large h.

4.9 MEQ Problem with Larger n

Our construction in Sections 4.6 and 4.7 can be generalized to larger networks. For brevity, just consider the case when $n = 3^m$. The nodes are organized in $m-1$ layers of "triangles". At the bottom ($(m-1)$-th) layer, there are 3^{m-1} triangles, each of which is formed with 3 nodes running the MEQ-AD(3,K) protocol presented in section 4.7. Then the i-th layer ($i < m-1$) consists 3^i triangles, each of which is formed with 3 "smaller" triangles from the $(i+1)$-th layer running the MEQ-AD(3,K) protocol. So the top layer consists of one triangle. For $K = 2^k$, the cost of this protocol is approximately

$$\frac{n-1}{2}(0.92 \times 2k + 7.755) \approx 0.92(n-1)k \tag{4.12}$$

for large k. Notice that $(n-1)k$ is the upper bound from Section 4.4. So the improvement of a constant factor of 0.92 can also be achieved for larger networks.

4.10 Summary

In this chapter, we study the communication complexity problem of the multiparty equality function, under the point-to-point communication model. The point-to-point communication model changes the problem significantly compared with previously used broadcast communication models. We focus on static protocols in which the schedule of transmissions is independent of the inputs. We then introduce techniques to significantly reduce the space of protocols to be studied. We then study the MEQ-AD(3,6) problem and introduce an optimal static protocol that achieves the minimum communication cost among all static protocols that solve the problem. This protocol is then used as a building block for construction of efficient protocols for more general MEQ-AD problems. The problem of finding the communication complexity of the MEQ problem for arbitrary values of n and K is still open.

CHAPTER 5

WATCHDOG IN WIRELESS NETWORKS

In wireless ad hoc and sensor networks, paths between a source and destination pair are usually multihop, and data packets are relayed in several wireless hops from their source to their destination. This multihop nature makes the wireless networks subject to tampering attack: a compromised/misbehaving node can easily ruin data communications by dropping or corrupting packets it should forward.

The watchdog mechanism proposed in [20] is a monitoring method used for ad hoc and sensor networks, and is the basis of many misbehavior detection algorithms and trust or reputation systems. The basic idea of the watchdog mechanism is that nodes (called watchdogs) police their downstream neighbors locally using overheard messages in order to detect misbehavior. If a watchdog detects that a packet is not forwarded within a certain period or is forwarded but altered by its neighbor, it deems the neighbor as misbehaving. When the misbehavior rate for a node surpasses a certain threshold, the source is notified and subsequent packets are forwarded along routes that exclude that node [20].

The main challenge for most watchdog mechanisms is the unreliable wireless environment. Due to possible reasons such as channel fading, collision with other transmissions, or interference, even when the source node and the attacker are both within the communication range, the watchdog may not be able to overhear every transmission and therefore may be unable to determine whether there is an attack.

To mitigate the misbehavior of the malicious nodes, a watchdog mechanism must achieve the following two goals: (1) Malicious behavior in the network should be detected; and (2) The throughput under the detection mechanism should be comparable to the throughput without detection if there is no attack. These two goals seem to conflict. On one hand, more redundancy is required to improve the probability of detection. On the other hand, higher

throughput requires redundancy to be reduced.

In this work, we show that both goals can be achieved simultaneously by introducing error detection block coding to the watchdog mechanism. The main contributions of this work are as follows:

- We propose a computationally simple scheme that integrates source error detection coding and the watchdog mechanism. We show that by choosing the encoder properly, a misbehaving node will be detected with high probability while the throughput approaches optimal, even in the case when the watchdog can only overhear a fraction of the packets and an omniscient attacker, i.e., the attacker knows what encoder is being used and no secret is shared only between the source and destination.

- We also propose a simple protocol that identifies the misbehaving node using exactly two watchdog nodes per unreliable relay node. We show that our protocol can be interpreted as a maximum likelihood decision making scheme. Finally, we show that with multiple rounds of detection, the probability of correctly locating the malicious node can be made arbitrarily close to one.

- We illustrated the effectiveness of our schemes with some small example topologies, and we also show that these results generalize to multihop networks.

5.1 Related Work

To ensure the reliability of packet delivery, trust for ad hoc and sensor networks has been investigated in past literature. The foundation of such dynamic trust systems is the node behavior monitoring mechanism, most frequent discussion being on the watchdog mechanism [20]. The main idea of watchdog was promiscuous monitoring, as discussed before. Once a node is deemed to be misbehaving, the source would choose a new route free of misbehaving node with the aid of a "path-rater".

A variant of the watchdog mechanism is proposed in [62] where next-hop's behavior is measured with the local evaluation record, defined as a 2-tuple: packet ratio and byte ratio, forwarded by the next-hop neighbor. Local

evaluation records are broadcast to all neighbors. The trust level of a node is the combination of its local observation and the broadcast information. Trust level is inserted to the RREQ (Route REQuest). Route is selected in the similar way to AODV (Ad hoc On Demand Distance Vector) [63]. Although many ad hoc trust or reputation systems such as [64], [65] and [66] adopt different trust level calculation mechanisms, the basic processes are similar to [62], including monitoring, broadcasting local observation, combing the direct and indirect information into the final trust level.

Recently, the security issue in network coding systems has drawn much attention. Due to the *mixing* nature of network coding, such systems are subject to a severe security threat, known as a *pollution attack*, where attackers inject corrupted packets into the network.

Several solutions to address pollution attacks in intra-flow coding systems use carefully designed digital signatures [67], [68], [69], [70] or hash functions [71], [72], which allow intermediate nodes to verify the integrity of combined packets. Packets that fail the test will be dropped to save some bandwidth. Such cryptographic solutions largely rely on either the private key being kept secret from the adversary or the difficulty of reversing the hash function. Non-cryptographic solutions have also been proposed [51], [46]. [73] proposes two practical schemes to address pollution attacks against network coding in wireless mesh networks without requiring complex cryptographic functions and incur little overhead. Reference [74] studies the transmission overhead associated with the schemes in [69], [51], and [46].

Reference [75] and our earlier work [22] propose two similar monitoring schemes, independently. The two-hop wireless network investigated in [75] is similar to the single flow example in section IV of [22] and section 5.2.1 of this chapter. Both schemes introduce redundancy at the source of data to improve the detection at the monitoring node, in the form of a polynomial hash function and MDS (maximum distance separable) code, respectively. While [75] considers general network codes, we focus on a particular network code – forward and compare. Both works show that as the amount of redundancy increases, the probability that the malicious node being undetected approaches zero.

5.2 Detecting Misbehavior

In this work, we focus on multihop wireless networks in which data packets are transmitted from source to destination through multiple relay nodes. We assume no coding is performed on relaying nodes so that packets are forwarded as they are received at the relay nodes. In such a network, a node W can be assigned as a watchdog for a relay node R if W can overhear both incoming and outgoing transmissions to and from R. W's duty is to compare the two copies of a packet it overhears from both R and its upstream neighbor, and to report an attack to the source or destination if there is a mismatch.

We are interested in detecting tampering attacks: we want the source or destination to be able to detect if there are misbehaving nodes in the network sending corrupted data. Moreover, we will focus tampering attack detection under a single node failure adversary model, i.e., the adversary can compromise at most one node in the network except for the source(s) and destination(s). If a watchdog is misbehaving, the only way to attack is to report an attack even though all other nodes are well-behaving. This is a trivial case since the source/destination always knows some node is misbehaving upon receiving the report of attack from the misbehaving watchdog. So it is more interesting to look at the case when a relay node misbehaves.

Since the wireless broadcast channel is usually unreliable, a watchdog node may only be able to overhear a fraction of the transmissions to/from the node it is monitoring for reasons such as channel fading and interference. As a result, an adversary may be able to avoid being detected by the watchdog with high probability by keeping the fraction of packets it tampers lower than a certain threshold $Th_{watchdog}$. To overcome this drawback of watchdog mechanisms, we propose to integrate source coding with watchdogs: the source node encodes the data packets with some error detecting code and sends the coded packets through the multihop network with watchdogs. By applying error detecting codes, the destination can detect an attack during the decoding process with high probability if the fraction of packets tampered with by the adversary is lower than a certain threshold Th_{code}. Intuitively, if $Th_{watchdog} < Th_{code}$, even an *omniscient* adversary will be detected with high probability no matter how many packets it corrupts. Throughout this chapter, we assume the adversary to be *omniscient*, i.e., the adversary has

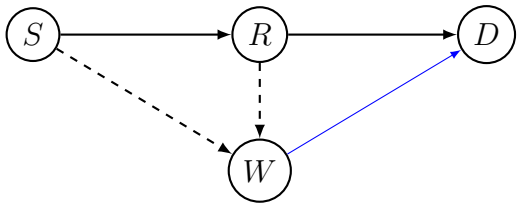


Figure 5.1: A single flow network. The thick (directed) lines denote a reliable connection from the tail node to the head node, a dashed line denotes the overhearing and a blue line denotes a secure asymptotically negligible rate channel between the two nodes.

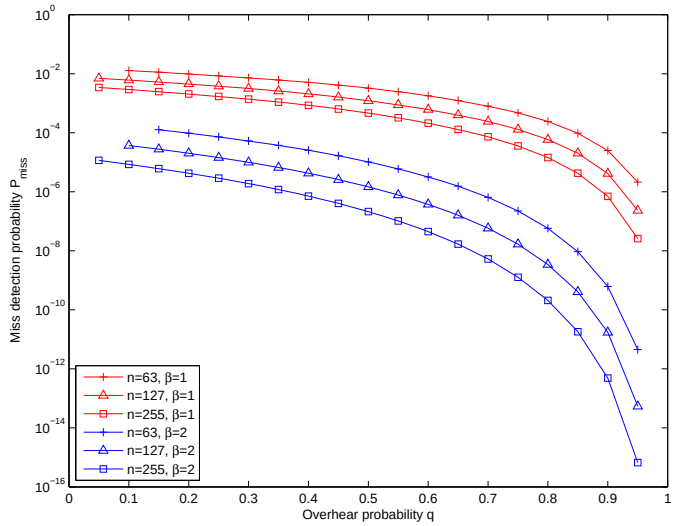
complete knowledge of the misbehaving detection mechanism being used, and there is no secret between the source and destination hidden from the adversary.

5.2.1 Single Flow Case

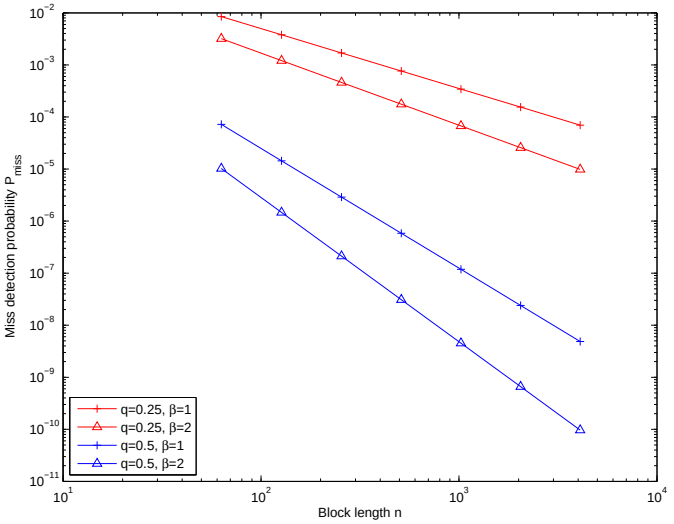
To illustrate the idea, let us look at the example of a single flow network as in Figure 5.1. There are 4 nodes in the network: the source node S, destination node D, attacker R, and the watchdog node W. The thick (directed) lines denote a link from the tail node to the head node, a dashed line denotes the overhearing and a blue line denotes a secure asymptotically negligible rate channel between the two nodes. We assume that all links (except for the blue one) have the same transmission rate of 1 packet per unit time. We also assume an optimal centralized schedule is enforced and the watchdog W knows what to compare. Moreover, we assume all transmissions along the path $S \to R \to D$ are reliable while W can only overhear both transmission of a packet with probability q [1].

The source node S encodes every k data packets into a block of n coded packets with an (n, k) MDS (maximum distance separable) code. We assume the packet size is large enough so that an MDS code always exists for the desired value of n and k. With an (n, k) MDS code, an attack will always be detected at the decoder as long as no more than $n - k$ packets are altered. As a result, R has to alter at least $n - k + 1$ packets in a block in order to

[1]Transmissions along the data path are usually protected by channel coding or/and retransmission mechanisms, while the watchdog can only overhear packets opportunistically.



(a) Miss detection probability v.s. observe probability



(b) Miss detection probability with $k = n + 1 - \frac{\beta \ln n}{q}$

Figure 5.2: Miss detection probability in the single flow example.

avoid being detected by the decoder. Since the more packets node R tampers with, the more likely it will be caught by node W, it is in R's interest to just attack the minimum number of packets per block: $n - k + 1$. In this case, it is easy to show that the probability of node R not being caught is

$$P_{miss}(n, k, q) = (1 - q)^{n-k+1}. \tag{5.1}$$

If we construct a (n, k) encoder such that

$$k = n + 1 - \frac{f(n,q)}{q}, \tag{5.2}$$

then from Equation 5.1 we have

$$P_{miss}(n, k, q) \leq e^{-q(n-k+1)} = e^{-f(n,q)}. \tag{5.3}$$

We can then choose the function $f(n, q)$ appropriately so that we can make P_{miss} arbitrarily small while the coding rate k/n approaches arbitrarily close to optimal (1). For example, by making $f(n, q) = \beta \ln n$ for any positive constant β, we have

$$\begin{aligned} P_{miss}(n, k, q) &\leq e^{-\beta \ln n} \\ &= n^{-\beta} \to 0 \ \text{ as } \ n \to \infty, \end{aligned} \tag{5.4}$$

and the coding rate becomes

$$\begin{aligned} \frac{k}{n} &= \frac{n + 1 - \frac{\beta \ln n}{q}}{n} \\ &= 1 + \frac{1}{n} - \frac{\beta}{q}\frac{\ln n}{n} \to 1 \ \text{ as } \ n \to \infty. \end{aligned} \tag{5.5}$$

So we can reduce the incentive for R to attack by making n large and choosing β appropriately.

Since the delay to verify a block equals the time it takes to transmit n packets in the block, the trade-off between probability of miss-detection and n is of interest. Figure 5.2(a) and Figure 5.2(b) show the probability of miss-detection with the observe probability q and with the number of packets n respectively. We can see that by integrating a watchdog and error detection coding, we can reduce the incentive for the attacker to attack by allowing longer delay.

Notice that by making n large, the coding/decoding complexity increases. In the case complexity is a concern, the source can scramble coded packets of multiple (n, k) encoded blocks and transmit these packets in a random order. By doing so, the attacker will have to corrupt more packets in order to destroy a particular block, which makes it easier to be detected by the

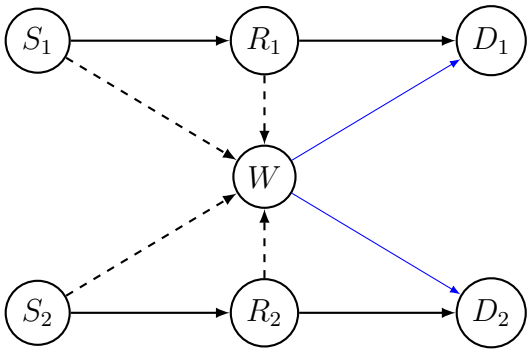


Figure 5.3: A two flow network.

watchdog.

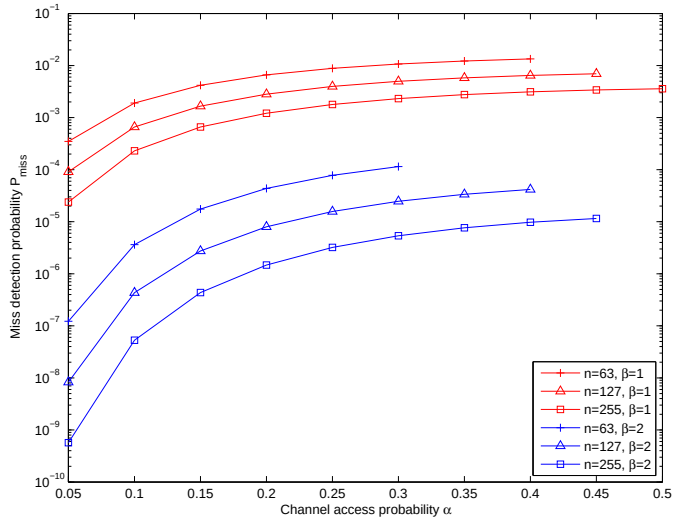
5.3 Two Flows Case

In Section 5.2.1, we have illustrated the effectiveness of source coding on top of watchdog mechanisms by a single flow example with a centralized optimal scheduler. In this section, we will study the trade-off between throughput and security in a more practical setting: there are multiple data flows in the network and a distributed random access MAC protocol is used. In the following example, we show that the proposed scheme achieves a high level of security while maintaining a reasonably good throughput.

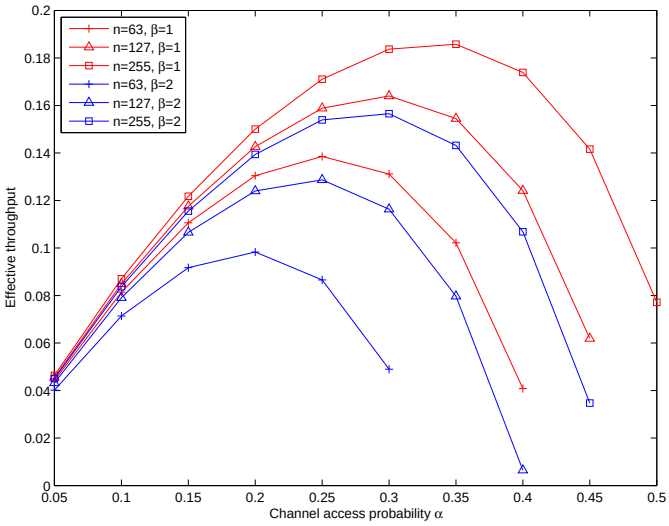
Consider the network shown in Figure 5.3 with two flows: $S_1 \to R_1 \to D_1$ and $S_2 \to R_2 \to D_2$. Suppose the flows are far enough away from each other so there is no inter-flow interference, but the watchdog W is sitting between the flows and can overhear transmissions on all the four links. So even though a transmission is successful along its path, it may collide with packets from the other flow received at W. We assume a slotted aloha access protocol with access probability α. To simplify the analysis, we further assume that a node will access the channel by transmitting dummy packets when it has no data packet to send. Under these assumptions, we can compute the throughput of each flow and the probability W can compare a particular packet as

$$T = \alpha(1-\alpha), \tag{5.6}$$

$$q = (1-\alpha)^5. \tag{5.7}$$



(a) Miss detection probability v.s. channel access probability



(b) Effective throughput v.s. channel access probability

Figure 5.4: Miss detection probability and effective throughput in the two flows example with $k = n + 1 - \frac{\beta \ln n}{(1-\alpha)^5}$. Where the curves stop means no code is available.

The exponent in Equation 5.7 is 5 because given that the transmission from S_1 to R_1 is successful, W can overhear it if neither S_2 nor R_2 transmits, which occurs with probability $(1-\alpha)^2$. To compare this packet, W should overhear the transmission from R_1 to D_1 too, which happens with probability $(1-\alpha)^3$ for S_1, S_2 and R_2 to remain silent.

Similar to the single-flow example, we can make P_{miss} arbitrarily small by

choosing

$$k = n + 1 - \frac{\beta \ln n}{(1-\alpha)^5}, \tag{5.8}$$

and the effective throughput is

$$\begin{aligned} T_E &= T \times \frac{k}{n} \\ &= \alpha(1-\alpha)(1+\frac{1}{n}) - \frac{\alpha\beta \ln n}{(1-\alpha)^4 n}. \end{aligned} \tag{5.9}$$

In Figure 5.4(a) and Figure 5.4(b), we plot the miss-detection probability and effective throughput when the error detection code is chosen according to Equation 5.8. We only plot the result for $\alpha \leq 0.5$ because further increasing α will only reduce the throughput. We can see from Figure 5.4(a) the probability of miss-detection increases as α increases and converges to roughly $n^{-\beta}$. Since the higher the α is, the fewer packets the watchdog can observe, the source has to sacrifice coding rate in order to maintain a certain probability of missing an attack as α increases.

As shown in Figure 5.4(b), as α increases, the effective throughput increases up to a certain level then drops to zero as α gets larger. We can also see the optimal access probability changes according to the value of n and β: the larger n, the higher α should be; the larger β, the smaller α should be. For instance, if the source does not perform any coding (which is not plotted here), it is well known that the optimal $\alpha = 0.5$ and the per-flow throughput is 0.25 packet per slot. In the case $n = 255$ and $\beta = 1$, the optimal α is about 0.35 and the throughput is about 0.19 packets per slot. Although the throughput is higher without source coding, it comes with the cost of not being able to provide any security guarantee. On the contrary, our scheme guarantees by upper-bounding P_{miss} by $n^{-\beta}$. Our scheme provides a method to optimize the balance among throughput, delay, and security.

5.4 Identifying the Misbehaving Node

In the previous section, we have studied the detection of misbehavior in the network. While misbehavior detection is essential in some applications, it is also important to identify the node that is misbehaving in order to avoid that node in future transmissions. The scheme discussed in the previous section

cannot determine which node is misbehaving.

In this section, we present a simple protocol that identifies the misbehaving node with two watchdogs. This includes the cases when a watchdog node is misbehaving. However, we show that for the proposed protocol, the adversary has no incentive to attack the watchdog. In particular, if the adversary attacks the watchdog, our protocol locates the adversarial node deterministically (with probability equal to one). However, if the adversary attacks the relay node, our scheme is guaranteed to locate the attacker with a probability that quickly approaches to unity with increasing number of packets transmitted.

The protocol in the following subsection can be viewed as several nodes making a decision on the correctness of the message transmitted by the relay node. The protocol can be visualized as the maximum likelihood decision scheme, and as we show in the following subsection, gives an optimal decision based on the decisions of the watchdogs.

5.4.1 The Protocol

Consider a relay node R that is observed by two watchdogs W_1 and W_2 and relays the information from a source node S to destination node D. Assume that the source node employs an (n, k)-MDS code. Assume that each source packet contains a unique *generation number* that identifies the generation to which a particular packet belongs. Each watchdog in the network decides whether or not the relay node is misbehaving based on all the overheard packets that belong to the current generation. If R is misbehaving (one of the n packets transmitted by R does not match the corresponding packet transmitted by S), it transmits a "decision bit" 1 to the judge node[2], else it transmits a decision bit 0 to the judge node. We assume that if the watchdog is misbehaving, it may transmit a 0 or a 1 for any particular relay node (same watchdog may transmit different decisions for different relay nodes). Denote the bits received from W_1 and W_2 by w_1 and w_2. The judge node collects the decision bits and makes a decision as follows:

[2]A judge node may be a destination node or the source node or both the nodes. In case of the destination node, it may decide to treat the information as authentic if it infers the relay node of not misbehaving. In case of the source node, it may decide to consider the path $S \to R \to D$ secure if it infers the relay node to be not misbehaving.

- $w_1w_2 = 11$: R is misbehaving;
- $w_1w_2 = 10$: W_1 is misbehaving;
- $w_1w_2 = 01$: W_2 is misbehaving;
- $w_1w_2 = 00$: none of the nodes is under attack.

We remark that our scheme gives a decision based on maximum likelihood probability of a particular node misbehaving. To see the protocol as a maximum likelihood decision making scheme, first consider the two simple cases of the decision bits being 11 and 00: in the former, the relay node must be misbehaving, else W_1 and W_2 cannot both detect a misbehavior at R (note that one of them can; if that particular watchdog is misbehaving, it could pretend that the relay node is actually misbehaving). And in the latter, there is no way to detect which node is misbehaving; indeed there may be no misbehaving node in such a case. For the case of 01 (10), note that if the attacker is at $W_1(W_2)$, $W_2(W_1)$ will never send a 1. Hence, assuming each node can be misbehaving with equal probability and the miss-detection probability for W_1 and W_2 are both P_{miss}, it is easy to compute probability of each node misbehaves given $w_1w_2 = 01$ as:

$$\begin{aligned} P_{W_1|01} &= 0 \\ P_{R|01} &= \frac{P_{miss} \times (1 - P_{miss})}{1 + (P_{miss} \times (1 - P_{miss}))} \\ P_{W_2|01} &= \frac{1}{1 + (P_{miss} \times (1 - P_{miss}))} \end{aligned}$$

The protocol in such a scenario decides that the watchdog sending a 1 is under attack, which is precisely the maximum likelihood decision given such a configuration (note that $P_{W_2|01} > P_{R|01}$).

We show in the following subsections that the misbehaving node can be located with a very high probability using just two watchdogs. We finally comment on how to bring the probability of correct location detection arbitrarily close to unity.

Let $P_{L|N}$ denote the probability of correctly locating the misbehaving node in the network given the adversary is at node N (where N may be R, W_1, or W_2); $P_{F|N}$ denote the probability that a node other than N is accused

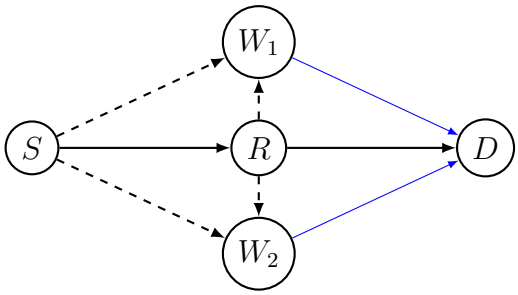


Figure 5.5: Single flow network of Figure 5.1 with an extra watchdogs.

to be misbehaving while in fact N is the adversary; and $P_{U|N}$ denote the probability when the adversary at node N operates undetected.

5.4.2 Performance – Single Flow Case

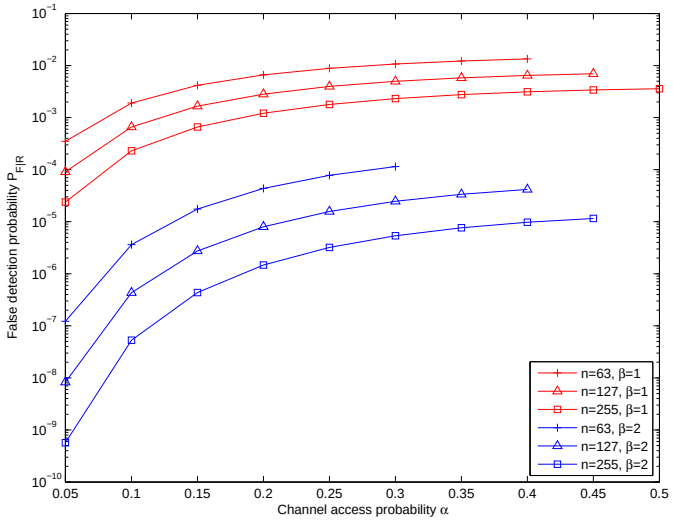
For the single flow case, only one extra watchdog is required to locate the adversary in the network (see Figure 5.5). We employ the protocol discussed above at destination D. Given this scheme, we have the following lemmas characterizing the performance of the protocol:

Lemma 5.1 *In single flow case of Figure 5.5, if any of the watchdogs is misbehaving, it will be located, i.e.,*

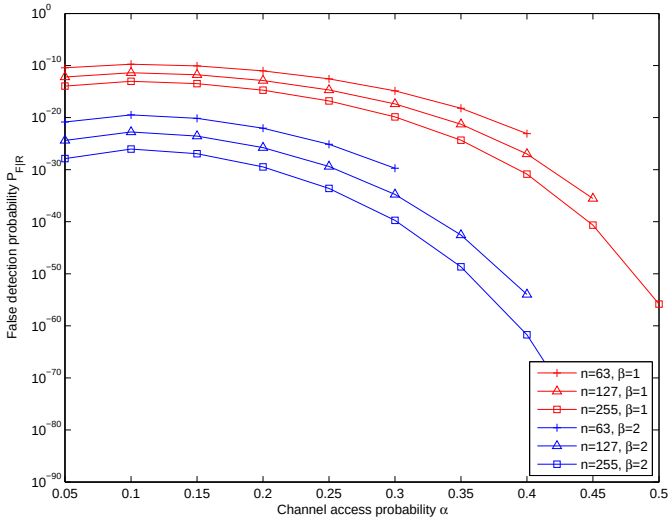
$$\begin{aligned} P_{L|W_1} &= P_{L|W_2} = 1 \\ P_{F|W_1} &= P_{U|W_1} = P_{F|W_2} = P_{U|W_2} = 0 \end{aligned}$$

Proof: Let us assume, without loss of the generality, that W_1 is misbehaving. In such a scenario, W_2 will always send a decision bit 0 to D since it will never overhear any incorrect packet being transmitted by R. A misbehaving W_1, on the other hand, will accuse the relay node of misbehaving. Then, the received decision bits at node D are 10. Given our protocol, D will decide that R is a reliable node and hence, the node W_1 sending a 1 must be misbehaving. Hence, D will always be able to locate the misbehaving node. □

The above lemma implies that the adversary has no incentive to attack either of the watchdogs in the network. Using the results of previous sections, this further restricts the capabilities of the attacker: it is not only restricted



(a) False detection probability v.s. channel access probability



(b) Miss-detection probability v.s. channel access probability

Figure 5.6: False location probability and undetected probability in the single flow example with $k = n + 1 - \frac{\beta \ln n}{(1-\alpha)^5}$. The curves stop where no code is available.

to attack the relay node but also needs to corrupt a large number of packets. The following lemma characterizes the performance of the protocol when the relay node misbehaves (corrupts more than $(n-k)$ packets out of n packets):

Lemma 5.2 *In single flow network of Figure 5.5, if R is misbehaving, then:*

$$\begin{aligned} P_{L|R} &= (1 - P_{miss})^2 \\ P_{F|R} &= 2 \times P_{miss} \times (1 - P_{miss}) \\ P_{U|R} &= P_{miss}^2. \end{aligned}$$

Proof: Let R be misbehaving and the decision bits sent by W_1 and W_2 be w_1 and w_2 respectively. Then, R goes undetected if and only if $w_1w_2 = 00$, i.e., when both the watchdogs miss all the packets corrupted by the attacker. Hence, the probability of R operating undetected is $P_{U|R} = P_{miss} \times P_{miss}$. On the other hand, R will be detected if and only if none of the watchdogs miss any of the packets corrupted by R, i.e., $w_1w_2 = 11$, leading to the fact that $P_{L|R} = (1 - P_{miss}) \times (1 - P_{miss})$.

Finally, the case of false detection is when exactly one of the watchdogs misses all the packets corrupted by R, i.e., when w_1w_2 is either 10 or 01; in this case W_1 or W_2 is detected as bad (not R). This gives $P_{F|R} = P_{miss} \times (1 - P_{miss}) + P_{miss} \times (1 - P_{miss})$. Notice that $P_{F|R} = 1 - (P_{L|R} + P_{U|R})$.

□

The probabilities $P_{F|R}$ and $P_{U|R}$ are plotted in Figure 5.6(a) and Figure 5.6(b) as a function of channel access probability for $k = n + 1 - \frac{\beta \ln n}{(1-\alpha)^5}$.

In Lemma 5.2, we have assumed that both the watchdogs have the same probability P_{miss}. This might not be the case since different nodes might observe different channel conditions due to being at different locations. We consider this case in the following subsection but the results of Lemma 5.2 can be modified easily to incorporate such a difference in probability of W_1 and W_2 missing the detection of packet modification by the relay node.

5.4.3 Performance – Two Flows Case

In this section, we study the location detection of the misbehaving node for the two flow case of Section 5.3. We first consider the case when the destination nodes may collaborate among themselves to locate the misbehaving node and show that such a collaboration does not necessarily reduce the connectivity requirement and/or improve the detection probability as long as the misbehaving node is not oblivious to the attack detection mechanism. We

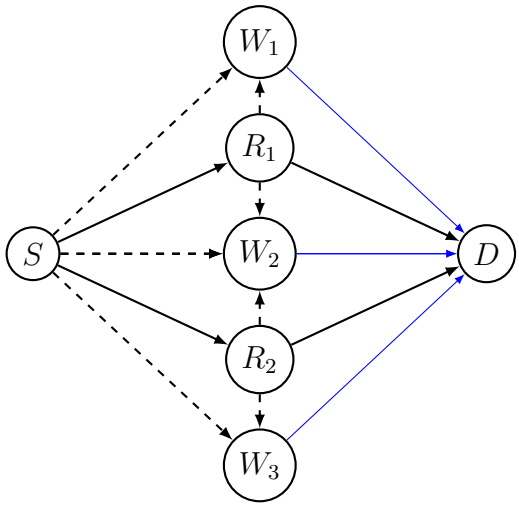


Figure 5.7: Corresponding netowrk for the two flow network of Figure 5.3 with extra watchdogs, when the judge nodes collaborate among themselves. The illustration also captures the multipath routing case when S relays the information to D via multiple relay nodes.

then show that the case of two flow network reduces to the case of multiple single flows with appropriate modifications to the probabilities of missing an attack at the watchdog nodes.

Assume that the two destinations D_1 and D_2 collaborate among themselves (share a few bits in order to locate the misbehaving node) and that the misbehaving node is oblivious to any attack detection mechanism in the network. This means that if the watchdog W_2 is the misbehaving node, it will send decision bits 1 to both D_1 and D_2. However, since there is a single adversary in the network, R_1 and R_2 cannot be both misbehaving. If D_1 and D_2 both receive 1 from W_2, they will (collaboratively) decide that W_2 is the misbehaving node. On the other hand, if R_1 or R_2 is misbehaving, W_2 sends a 1 to the corresponding destination node and a 0 to the other destination node, which will certainly imply that the corresponding relay node is under attack (assuming that W_2 is oblivious to the attack detection mechanism).

Notice that in the above case, we do not need W_1 and W_3 for locating the misbehaving node. The problem arises when the misbehaving node knows that an attack detection scheme is being employed in the network. In such a case, the misbehaving node (at W_2) may send a decision bit 1 to one destination node (say D_1) and a 0 to the other destination node, making D_1 (incorrectly) think that R_1 is actually misbehaving. In such a case, we

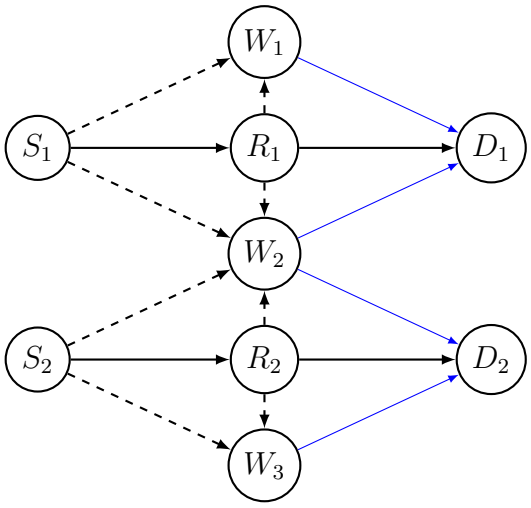


Figure 5.8: Two flow network of Figure 5.3 with extra watchdogs.

need W_1 and W_3 to be able to correctly decide the location of the adversary. Note that the above discussion implies that even if several judge nodes start collaborating, at least two watchdogs are required to correctly locate the misbehaving node. Hence, collaboration of judge nodes does not help in reducing connectivity requirements and/or devising a better attack detection scheme.

Notice that the above discussion of collaborating judge nodes also captures the multipath transmission mechanism where a source node might relay the information to the same destination via multiple relay nodes (see Figure 5.7). Hence, to (correctly) locate the misbehaving node, the connectivity requirements for the network is every relay node being monitored by at least two watchdogs. We derive the results for the two flow case when the judge nodes do not collaborate but as discussed above, these results hold even if the judge nodes collaborate among themselves.

If the destination nodes do not collaborate, then the decision made by any of the destination nodes, say D_1, is dependent only on the decision bits of the watchdogs observing the corresponding relay node, i.e., W_1 and W_2 for D_1 (similar remarks hold for D_2). This in turn means that each destination node individually behaves as if it is participating in a single flow network. However, as discussed earlier, it might be the case that the watchdogs W_1 and W_3 have probabilities of detection different from that of W_2. The following lemmas hold for the case of two flow network of Figure 5.8, where we denote

the probabilities of missing an attack at the relay node for watchdogs W_1 and W_3 are $P_{miss,1}$ and that of W_2 is $P_{miss,2}$.

Lemma 5.3 *In the two flow case of Figure 5.8 with our protocol, if the attacker attacks at any of the watchdogs, it will be located, i.e.,*

$$\begin{aligned} P_{L|W_1} &= P_{L|W_2} = 1 \\ P_{F|W_1} &= P_{U|W_1} = P_{F|W_2} = P_{U|W_2} = 0 \end{aligned}$$

Proof: Similar to Lemma 5.1, collaboration of destination nodes does not play a role.

□

Lemma 5.4 *In the two flow case with our protocol, if the adversary attacks R_1 or R_2, then:*

$$\begin{aligned} P_{L|R_1} &= P_{L|R_2} = (1 - P_{miss,1}) \times (1 - P_{miss,2}) \\ P_{F|R_1} &= P_{F|R_2} = P_{miss,1} + P_{miss,2} - 2 \times P_{miss,1} P_{miss,2} \\ P_{U|R_1} &= P_{U|R_2} = P_{miss,1} \times P_{miss,2} \end{aligned}$$

Proof: Similar to Lemma 5.2, collaboration of destination nodes does not play a role.

□

5.5 Summary

In this work, we have studied the problem of misbehavior detection in wireless networks. We propose a lightweight misbehavior detection scheme which integrates the idea of watchdogs and error detection coding. We show in a single flow example that even if the watchdog can only observe a fraction of packets, by choosing the encoder properly, an attacker will be detected with high probability while achieving throughput arbitrarily close to optimal. The trade-off between throughput and security in a more practical setting – there are multiple data flows in the network and a distributed random access MAC protocol is used – is also studied. This technique can also be used to locate the misbehaving node, by using just one extra watchdog per relay node.

CHAPTER 6

FUTURE WORK

In this chapter, we summarize the future goals of our research.

6.1 Fast Oblivious Byzantine Agreement Algorithms

In the Byzantine broadcast algorithms we present in Chapter 3, every time a failure is detected, the dispute control (or fault diagnosis) broadcast stage is performed in addition to the normal operations. Moreover, the order according to which the nodes transmit will change after a mode transition, depending on the location of the failure. In practice, the diagnostic process is preferred to be avoided since it introduces excessive delay before agreement is achieved for the current instance. It is also desirable that the algorithm is *oblivious*, by which we mean the schedule of transmissions does not change over time. *Fast oblivious* algorithms that have these properties is one possible topic for future work.

6.2 Byzantine Agreement with Capacity Constraints in Wired Networks

In Chapter 3, we have presented the NAB algorithm that achieves at least 1/3 of the capacity of Byzantine broadcast for general point-to-point networks. For future work, providing better performance guarantee in the following two aspects are of interests:

1. Provide tighter upper bounds on the capacity of Byzantine broadcast or consensus in general point-to-point networks.

2. Develop Byzantine broadcast or consensus algorithm that achieves better throughput than NAB. One possible approach for achieving this

goal is to combine the unreliable broadcast in Phase 1 and fault detection in Phase 2 of NAB. The two capacity-achieving Byzantine broadcast algorithms for four-node networks and symmetric networks suggest good potential of this approach.

6.3 Byzantine Agreement in Wireless Networks

In Chapter 5, we have explored the broadcast property of the wireless medium for misbehavior detection in unicast communication. In fact, Byzantine agreement algorithms may also benefit from the broadcast communication model when wireless medium is considered. Therefore, investigating the Byzantine agreement problem in the broadcast communication model setting is part of the future work. A potential strategy would be try to integrate the watchdog mechanism into a Byzantine agreement algorithm.

6.4 Multiparty Equality Function Computation with Capacity Constraints

As we have seen in the previous chapters on Byzantine agreement, the main procedure of our agreement algorithms is to check whether all $n-f$ fault-free nodes decide on the same value, which is in fact solving the MEQ problem for every subset of $n-f$ nodes in the network. In Chapter 4, preliminary results of the MEQ problem with no capacity constraint have been discussed. However, even the smallest non-trivial instance of the problem – MEQ$(3, K)$ – is still unsolved for general K. Part of the future work would be to continue investigating the MEQ$(3, K)$ problem. The first step would be to study the MEQ$(3, K)$ problem under point-to-point capacity constraints, which we believe would provide useful insight for the unconstrained problem. Meanwhile, better understanding of the MEQ problem in the constrained setting will likely provide useful insights for the Byzantine agreement problem.

CHAPTER 7

CONCLUSIONS

In this dissertation, we have discussed various algorithms for tolerating failures in distributed systems. More specifically, in Chapter 2, we propose CBC, a linear-complexity Byzantine consensus algorithm; and CBB, a linear-complexity Byzantine broadcast algorithm. In design of both CBC and CBB, we incorporate techniques such as error-detection coding and system level diagnosis to improve the communication complexity of Byzantine consensus and broadcast, without relying on any cryptographic assumption.

In Chapter 3, we study the design of network-aware Byzantine agreement algorithms. In particular, we focus on optimizing the throughput of Byzantine broadcast in point-to-point networks, in which each link has a link capacity constraint. We derive an upper bound on the achievable throughputs of Byzantine broadcast for general point-to-point networks. A network-aware Byzantine broadcast algorithm NAB is proposed and proved to achieve throughput at least 1/3 fraction of the optimal in general point-to-point networks. In two particular families of point-to-point networks, capacity-achieving Byzantine broadcast algorithms are developed.

In Chapter 4, we investigate the problem of multiparty equality (MEQ) function computation, under the point-to-point communication model. The point-to-point communication model changes the problem significantly compared with previously used broadcast communication models. We focus on static protocols in which the schedule of transmissions is independent of the inputs. We then introduce techniques to significantly reduce the space of protocols to be studied. We then study the MEQ-AD(3,6) problem and introduce an optimal static protocol that achieves the minimum communication cost among all static protocols that solve the problem. This protocol is then used as a building block for construction of efficient protocols for

more general MEQ-AD problems. The problem of finding the communication complexity of the MEQ problem for arbitrary values of n and K is still open.

In Chapter 5, we study the problem of misbehavior detection in wireless networks. We propose a lightweight misbehavior detection scheme which integrates the idea of watchdogs and error-detection coding. We show in a single flow example that even if the watchdog can only observe a fraction of packets, by choosing the encoder properly, an attacker will be detected with high probability while achieving throughput arbitrarily close to optimal. The trade-off between throughput and security in a more practical setting – there are multiple data flows in the network and a distributed random access MAC protocol is used – is also studied. This technique can also be used to locate the misbehaving node, by using just one extra watchdog per relay node.

The design principle of the fault-tolerant algorithms in this dissertation is based on the observation that if the locations of the faulty component(s) in the system do not change frequently, error-detection suffices, in contrast to the commonly adopted error-correction approach. Error-detection guarantees safety, i.e., no erroneous outcome can be arrived. When error (or failure) is detected, diagnostic operations can be performed to learn information of the locations of the faulty components and reduce their capability in interfering with the normal operation of the algorithm in the future. It is our belief that this methodology could be applied to other open challenging problems in fault-tolerance networking and distributed computing research.

APPENDIX A

ERROR-FREE BYZANTINE FAULT TOLERANCE WITH LINEAR COMPLEXITY

A.1 Improving Computational Complexity of CBC

To make the algorithm computationally more efficient, we need to modify Algorithm 1 slightly, as elaborated later in this appendix. With this change, the algorithm only looks for a set P_{match} of size $n - f$ such that all the fault-free processors in $P_{match} \cap P_{good}$ have the same input in generation g. The algorithm's response when such a P_{match} is not found is now somewhat different, as sketched below. A complete description and the proof of correctness of the modified algorithm is omitted for brevity.

P_{match} is found as follows. We maintain a set Q that contains the largest set of processors that appear to have identical input up to the previous generation. Initially, Q is the set of all n processors. The matching stage is performed as it is in Algorithm 1, up to Line 3(d). The subsequent steps of the matching stage are different.

(i) Determine the largest set $Q' \subseteq Q$ such that all the processors in set Q' have M vectors that contain at least $n-f$ **TRUE** entries. If $|Q'| < n-f$, then the fault-free processors must have different L-bit inputs, and the algorithm terminates with the decision being a default value. If $|Q'| \geq n - f$, the proceed to the following steps.

(ii) For every pair of nodes $i, j \in Q'$ that trusts each other, if there are more than t distinct processors not trusted by node i or j, then one of nodes i and j must be faulty. Remove edge (i, j) in the diagnosis graph, set $M_i[j] =$ **FALSE** and $M_j[i] =$ **FALSE**, and go back to step (i) above.

(iii) For every pair of nodes $i, j \in Q'$ (that now trust the same set of at least $n - f$ processors), check whether $M_i[k] = M_j[k]$ for each node k that is trusted by both nodes i and j. If this check fails, then either $v_i(g)$ and $v_j(g)$ are different (or, pretending to be different, in case one of these processors is

faulty), or node k has sent different symbols to nodes i and j. In this case, go to step (iv); otherwise it can be proved that $v_i(g) = v_j(g)$ if nodes i and j are both fault-free. If all these checks pass, then P_{match} can be chosen as any subset of Q' of size $n - f$, and it always contains a clique of at least $n - 2t$ fault-free processors that have the same input for the current generation. Then proceed to the Checking stage as in Algorithm 1.

(iv) If misbehavior, or difference in processor inputs, is detected in step (iii) above, some additional steps are needed: All processors in Q' broadcast their inputs for generation g. Q is then updated as the largest subset of Q' that broadcast the same value. If $|Q| < n - f$, then terminate and decide on a default output. If $|Q| \geq n - f$, then decide on the value broadcast by processors in Q. Additionally, diagnosis is also performed to remove an edge from the diagnosis graph, if misbehavior has indeed occurred.

A.2 Proof of Correctness of Improved-CBC

In this section, we prove the correctness of Algorithm 2.6.2. Similar to the proof of CBC, in the proofs of the following lemmas, we assume that the fault-free nodes always trust each other.

Lemma A.1 *If $Detected_j$ =***FALSE** *for every node j in Line 2(d), every fault-free node $i \in P_{good}$ decides on the identical output value $y(g)$ such that $y(g) = x_j(g)$ for every node $j \in P_{good} \cap P_{match}$.*

Proof: According to the algorithm, every fault-free node $i \in P_{good}$ has sent $S_i[i]$ (computed from $x_i(g)$ directly if node $i \in P_{match}$, or computed using symbols received in Lines 1(a) and 1(b) if node $i \notin P_{match}$) to all the other fault-free nodes. As a result, $R_i|P_{good} = R_j|P_{good}$ is true for every pair of fault-free nodes $i, j \in P_{good}$. Since $|P_{good}| \geq n - f$ and C_{n-f} is a distance-$(f + 1)$ code, it follows that either all fault-free nodes in P_{good} decide on the same output, or at least one fault-free node $i \in P_{good}$ sets $Detected_i \leftarrow$**TRUE** in Line 2(a). In the case all $Detected_j$=**FALSE**, all fault-free nodes decide on an identical $y(g)$. Moreover, according to Line 2(b), every fault-free node $j \in P_{good} \cap P_{match}$ finds $R_j = S_j$. It then follows that $y(g) = C_{n-f}^{-1}(R_j) = C_{n-f}^{-1}(S_j) = x_j(g)$.

□

Lemma A.2 *If a P_{new} such that $|P_{new}| \geq n-f$ is found in Line 3(g), every fault-free node $i \in P_{good}$ decides on an identical output value $y(g)$ such that $y(g) = x_j(g)$ for every node $j \in P_{good} \cap P_{new}$.*

Proof: Since $|P_{new}| \geq n-f$ and since at most f nodes are faulty, there must be at least $n-2f$ fault-free nodes in $P_{good} \cap P_{new}$, which have broadcast the same $S^{\#}$'s in Line 3(b). So at Line 3(h), all fault-free nodes decide on the identical output $y(g) = x_j(g)$ for node $j \in P_{good} \cap P_{new}$.

□

Lemma A.3 *If a P_{new} such that $|P_{new}| \geq n-f$ can not be found in Line 3(g), then there must be two fault-free nodes $i, j \in P_{good}$ such that $x_i \neq x_j$.*

Proof: It is easy to see that if all fault-free nodes in P_{good} are given the same input, then a P_{new} such that $|P_{new}| \geq n-f$ can always be found in Line 3(g). Then the lemma follows.

□

The correctness of the way Diag_Graph is updated is proved in the same way as in CBC. Now we can conclude the correctness of Algorithm Improved-CBC as the following theorem:

Theorem A.1 *Given n nodes with at most $f < n/3$ being faulty, each given an input value of L bits, Algorithm 2.6.2 achieves consensus correctly in L/D generations, with the diagnosis stage performed for at most $f+f(f+1)$ times.*

Proof: According to Lemmas A.1 and A.2, the decided output $y(g)$ always equals to $x_j(g)$ for some node $j \in P_{good} \cap P_{match}$, unless $|P_{new}| < n-f$ in Line 3(h). So consistency and validity properties are satisfied until $|P_{new}|$ becomes $< n-f$. In the case $|P_{new}| < n-f$, according to Lemma A.3, there must be two fault-free nodes that are given different inputs. Then it is safe to decide on a default output and terminate. So the L-bit output satisfies the consistency and validity properties.

Every time the diagnosis stage is performed, either at least one edge in Diag_Graph associated with a faulty node is removed, or at least one node is removed from P_{match}. So it takes at most $f(f+1)$ instances of the diagnosis stage before all faulty nodes are identified. In addition, it will take at most f instances to remove fault-free nodes from P_{match} until two fault-free nodes

are identified as having different inputs, and the algorithm terminates with a default output. □

A.3 Proof of Correctness of VCBC

Lemma A.4 *If there are a set of at least v fault-free nodes $Q \subseteq P_{good}$ such that for each node $i \in Q$, $x_i(g) = x(g)$ for some $x(g)$, then a set P_{match} of size v necessarily exists.*

Proof: Since all the fault-free nodes in Q have identical input $x(g)$, $S_i = C_v(x(g))$ for every node $i \in Q$. Since these nodes are fault-free and always trust each other, they send each other correct messages in the matching stage. Thus, $R_i[j] = S_j[j] = S_i[j]$ for every pair of nodes $i, j \in Q$. This fact implies that $M_i[j] = M_j[i] =$**TRUE** for all pairs $i, j \in Q$. Since there are $|Q| \geq v$ fault-free nodes in Q, it follows that a set P_{match} of size v must exist. □

Lemma A.5 *If $Detected_j =$***FALSE** *for every node j in Line 2(d), every fault-free node $i \in P_{good}$ decides on the identical output value $y(g)$ such that $y(g) = x_j(g)$ for every node $j \in P_{match} \cap P_{good}$.*

Proof: Observe that size of set $P_{match} \cap P_{good}$ is at least $v - f \geq 1$, so there must be at least one fault-free node in P_{match}.

According to the algorithm, every fault-free node $i \in P_{good}$ has sent $S_i[i]$ (computed from $x_i(g)$ directly if node $i \in P_{match}$, or computed using the v symbols received from P_{match} in Lines 1(a) and 1(f) if node $i \notin P_{match}$) to all the other fault-free nodes. As a result, $R_i|P_{good} = R_j|P_{good}$ is true for every pair of fault-free nodes $i, j \in P_{good}$. Since $|P_{good}| \geq n - f \geq v$ and C_v has dimension v, it follows that either all fault-free nodes P_{good} decide on the same output, or at least one fault-free node $i \in P_{good}$ sets $Detected_i \leftarrow$**TRUE** in Line 2(a). In the case $Detected_j =$**FALSE** for every node j, all fault-free nodes decide on an identical $y(g)$. Moreover, according to Line 2(b), every fault-free node $j \in P_{good} \cap P_{match}$ finds $R_j = S_j$. It then follows that $y(g) = C_v^{-1}(R_j) = C_v^{-1}(S_j) = x_j(g)$ where node $j \in P_{good} \cap P_{match}$.

□

Then we can have the following theorem about the correctness of Algorithm 2.6.3.

Theorem A.2 *Given n nodes with at most $f < n/3$ being faulty, each given an input value of L bits, Algorithm 2.6.3 achieves v-validity for each one of the L/D generations, with the diagnosis stage performed for at most $f(f+1)$ times.*

Proof: Similar to Theorem A.1. □

A.4 Proof of Theorem 2.2

Proof: First consider the case when source node n is faulty. In this case, every fault-free peer $i < n$ sends $R_i[i]$ to all peers that it trusts. Also, as we have discussed in Section 2.7.2, fault-free node always trust each other. These two facts together imply that among coded symbols received by the fault-free peers, at least $n - f$ of them are shared identically. According to the property of the $(2(n-1), n-f)$ Reed-Solomon code C_{n-f}, either all fault-free peers succeed in decoding the received symbols to some identical output, or at least one of them cannot decode and detects the failure.

In the case that source node n is fault-free, each fault-free peer i receives at least $n - f$ coded symbols either directly from node n or through other fault-free peers. So these symbols must be the same as the corresponding ones in $C_{n-f}(x(g))$, and hence $C_{n-f}^{-1}(R_i) = x(g)$ if it succeeds. □

APPENDIX B

NETWORK-AWARE BYZANTINE AGREEMENT ALGORITHM DESIGN

B.1 Unreliable Broadcast in Phase 1

According to [76], in a given graph $\mathcal{G}_k$ with $\gamma_k = \min_{j \in \mathcal{V}_k} MINCUT(\mathcal{G}_k, 1, j)$, there always exist a set of γ_k unit-capacity spanning trees of $\mathcal{G}_k$ such that the total usage on each edge $e \in \mathcal{E}_k$ by all the γ_k spanning trees combined is no more than its link capacity z_e. Each spanning tree is "unit-capacity" in the sense that 1 unit capacity of each link on that tree is allocated for transmissions on that tree. For example, Figure 3.2(c) shows 2 unit-capacity spanning trees that can be embedded in the directed graph in Figure 3.2(a): one spanning tree is shown with solid edges and the other spanning tree is shown in dotted edges. Observe that link (1,2) is used by both spanning trees, each tree using a unit capacity on link (1,2), for a total usage of 2 units, which is the capacity of link (1,2).

To broadcast an L-bit value from source node 1, we represent the L-bit value as γ_k symbols, each symbol being represented using L/γ_k bits. One symbol (L/γ_k bits) is then transmitted along each of the γ_k unit-capacity spanning trees.

B.2 Dispute Control

The dispute control algorithm is performed in the k-th instance of NAB only if at least one node misbehaves during Phases 1 or 2. The goal of dispute control is to learn some information about the identity of at least one faulty node. In particular, the dispute control algorithm will identify a new node as being faulty, or/and identify a new node pair in dispute (at least one of the nodes in the pair is guaranteed to be faulty). The steps in dispute control in

the k-th instance of NAB are as follows:

- (DC1) Each node i in $\mathcal{V}_k$ uses a previously proposed Byzantine broadcast algorithm, such as [35], to broadcast to all other nodes in $\mathcal{V}_k$ all the messages that this node i claims to have received from other nodes, and sent to the other nodes, during Phases 1 and 2 of the k-th instance. Source node 1 also uses an existing Byzantine broadcast algorithm [35] to broadcast its L-bit input for the k-th instance to all the other nodes. Thus, at the end of this step, all the fault-free nodes will reach correct agreement for the output for the k-th instance.

- (DC2) If for some node pair $a, b \in \mathcal{V}_k$, a message that node a claims above to have sent to node b mismatches with the claim of received messages made by node b, then node pair a, b is found in dispute. In step DC1, since a Byzantine broadcast algorithm is used to disseminate the claims, all the fault-free nodes will identify identical node pairs in dispute.

  It should be clear that a pair of fault-free nodes will never be found in dispute with each other in this step.

- (DC3) The NAB algorithm is deterministic in nature. Therefore, the messages that should be sent by each node in Phases 1 and 2 can be completely determined by the messages that the node receives, and, in case of node 1, its initial input. Thus, if the claims of the messages sent by some node i are inconsistent with the message it claims to have received, and its initial input (in case of node 1), then that node i must be faulty. Again, all fault-free nodes identify these faulty nodes identically. Any nodes thus identified as faulty until now (including all previous instances of NAB) are deemed to be "in dispute" with all their neighbors (to whom the faulty nodes have incoming or outgoing links).

  It should be clear that a fault-free node will never be found to be faulty in this step.

- (DC4) Consider the node pairs that have been identified as being in dispute in DC2 and DC3 of at least one instances of NAB so far.

We will say that a set of nodes F_i, where $|F_i| \leq f$, "explains" all the disputes so far, if for each pair a, b found in dispute so far, at least one of a and b is in F_i. It should be easy to see that for any set of disputes that may be observed, there must be at least one such set that *explains* the disputes. It is easy to argue that the nodes in the set below must be necessarily faulty (in fact, the nodes in the set intersection below are also guaranteed to include nodes identified as faulty in step DC3).

$$\bigcap_{\delta=1}^{\Delta} F_\delta$$

Then, $\mathcal{V}_{k+1}$ is obtained as $\mathcal{V}_k - \bigcap_{\delta=1}^{\Delta} F_\delta$. $\mathcal{E}_{k+1}$ is obtained by removing from $\mathcal{E}_k$ edges incident on nodes in $\bigcap_{\delta=1}^{\Delta} F_\delta$, and also excluding edges between node pairs that have been found in dispute so far.

As noted earlier, the above dispute control phase may be executed in at most $f(f+1)$ instances of NAB.

B.3 Proof of Theorem 3.1

To prove Theorem 3.1, we first prove that when the coding matrices are generated at random as described, for a particular subgraph $H \in \Omega_k$, with non-zero probability, the coding matrices $\{\mathbf{C_e} | e \in \mathcal{G}_k\}$ define a matrix $\mathbf{C_H}$ (as elaborated later) such that $\mathbf{D_H C_H} = \mathbf{0}$ if and only if $\mathbf{D_H} = \mathbf{0}$. Then we prove that this is also **simultaneously** true for all subgraphs $H \in \Omega_k$.

B.3.1 For a Given Subgraph $H \in \Omega_k$

Consider any subgraph $H \in \Omega_k$. For each edge $e = (i, j)$ in H, we "expand" the corresponding coding matrix $\mathbf{C_e}$ (of size $\rho_k \times z_e$) to a $(n-f-1)\rho_k \times z_e$ matrix $\mathbf{B_e}$ as follows: $\mathbf{B_e}$ consists $n-f-1$ blocks, each block is a $\rho_k \times z_e$ matrix:

- If $i \neq n-f$ and $j \neq n-f$, then the i-th and j-th block equal to $\mathbf{C_e}$

and $-\mathbf{C_e}$, respectively. The other blocks are all set to $\mathbf{0}$.

$$\mathbf{B_e} = \begin{array}{r} \\ \\ \\ i\text{-th block} \\ \\ \\ \\ j\text{-th block} \\ \\ \\ \end{array} \begin{pmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{C_e} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ -\mathbf{C_e} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}.$$

- If $i = n - f$, then the j-th block equals to $-\mathbf{C_e}$, and the other blocks are all set to $\mathbf{0}$ matrix.

$$\mathbf{B_e} = \begin{array}{r} \\ \\ \\ j\text{-th block} \\ \\ \\ \end{array} \begin{pmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ -\mathbf{C_e} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}.$$

- If $j = n - f$, then the i-th block equals to $\mathbf{C_e}$, and the other blocks are

all set to $\mathbf{0}$ matrix.

$$\mathbf{B_e} = \ i\text{-th block} \begin{pmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{C_e} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}.$$

Let $D_i(\beta) = X_i(\beta) - X_{n-f}(\beta)$ for $i < n-f$ as the difference between $\mathbf{X_i}$ and $\mathbf{X_{n-f}}$ in the β-th element. Recall that $\mathbf{D_i} = \mathbf{X_i} - \mathbf{X_{n-f}} = \begin{pmatrix} D_i(1) & \cdots & D_i(\rho_k) \end{pmatrix}$ and $\mathbf{D_H} = \begin{pmatrix} \mathbf{D_1} & \cdots & \mathbf{D_{n-f-1}} \end{pmatrix}$. So $\mathbf{D_H}$ is a row vector of $(n-f-1)\rho_k$ elements from $GF(2^{L/\rho_k})$ that captures the differences between $\mathbf{X_i}$ and $\mathbf{X_{n-f}}$ for all $i < n-f$. It should be easy to see that, for edge $e = (i, j)$

$$(\mathbf{X_i} - \mathbf{X_j})\mathbf{C_e} = \mathbf{0} \Leftrightarrow \mathbf{D_H}\mathbf{B_e} = \mathbf{0}.$$

So for edge e, steps 1-2 of Algorithm 3.4.5 have the effect of checking whether or not $\mathbf{D_H}\mathbf{B_e} = \mathbf{0}$.

If we label the set of edges in H as $e1, e2, \cdots$, and let m be the sum of the capacities of all edges in H, then we construct a $(n-f-1)\rho_k \times m$ matrix $\mathbf{C_H}$ by concatenating all expanded coding matrices:

$$\mathbf{C_H} = \begin{pmatrix} \mathbf{B_{e1}} & \mathbf{B_{e2}} & \cdots \end{pmatrix}$$

where each column of $\mathbf{C_H}$ represents the equality checking of one coded symbol sent in H over the corresponding edge. Note that the sum of the capacities of all edges in H equals to m. Then steps 1-2 of Algorithm 3.4.5 for all edges in H have the same effect of checking whether or not $\mathbf{D_H}\mathbf{C_H} = \mathbf{0}$. So to prove Theorem 3.1, we need to show that there exists at least one $\mathbf{C_H}$ such that

$$\mathbf{D_H}\mathbf{C_H} = \mathbf{0} \ \Leftrightarrow \ \mathbf{D_H} = \mathbf{0}.$$

It is obvious that if $\mathbf{D_H} = \mathbf{0}$, then $\mathbf{D_H}\mathbf{C_H} = \mathbf{0}$ for any $\mathbf{C_H}$. So all left to show is that there exists at least one $\mathbf{C_H}$ such that $\mathbf{D_H}\mathbf{C_H} = \mathbf{0} \Rightarrow \mathbf{D_H} = \mathbf{0}$.

It is then sufficient to show that $\mathbf{C_H}$ contains a $(n-f-1)\rho_k \times (n-f-1)\rho_k$ submatrix $\mathbf{M_H}$ that is **invertible**, because when such an invertible submatrix exists,

$$\mathbf{D_H C_H = 0} \;\Rightarrow\; \mathbf{D_H M_H = 0} \;\Rightarrow\; \mathbf{D_H = 0}.$$

Now we describe how one such submatrix $\mathbf{M_H}$ can be obtained. Notice that each column of $\mathbf{C_H}$ represents one coded symbol sent on the corresponding edge. A $(n-f-1)\rho_k \times (n-f-1)$ submatrix $\mathbf{S}$ of $\mathbf{C_H}$ is said to be a "spanning matrix" of H if the edges corresponding to the columns of $\mathbf{S}$ form a undirected spanning tree of $\overline{H}$; recall that $\overline{H}$ is the *undirected* representation of H. In Figure 3.2(d), an undirected spanning tree of the undirected graph in Figure 3.2(b) is shown in dotted edges. It is worth pointing out that an undirected spanning tree in an undirected graph $\overline{H}$ does not necessarily correspond to a directed spanning tree in the corresponding directed graph H. For example, the directed edges in Figure 3.2(a) corresponding to the dotted undirected edges in Figure 3.2(d) do not form a spanning tree in the directed graph in Figure 3.2(a).

It is known that in an undirected graph whose MINCUT equals to U, at least $U/2$ undirected unit-capacity spanning trees can be embedded [76].[1] This implies that the sum of link capacities $m \geq (n-f-1)U_k/2$, and $\mathbf{C_H}$ contains a set of $U_k/2$ spanning matrices such that no two spanning matrices in the set covers the same column in $\mathbf{C_H}$. Let $\{\mathbf{S_1}, \cdots, \mathbf{S}_{\rho_k}\}$ be one set of $\rho_k \leq U_k/2$ such spanning matrices of H. Each of these spanning matrices has dimension $(n-f-1)\rho_k \times (n-f-1)$. Then the union of these spanning matrices forms an $(n-f-1)\rho_k \times (n-f-1)\rho_k$ submatrix of $\mathbf{C_H}$:

$$\mathbf{M_H} = \begin{pmatrix} \mathbf{S_1} & \cdots & \mathbf{S}_{\rho_k} \end{pmatrix}.$$

Next, we will show that when the set of coding matrices are generated as described in Theorem 3.1, with non-zero probability we obtain an invertible square matrix $\mathbf{M_H}$. When $\mathbf{M_H}$ is invertible,

$$\mathbf{D_H M_H = 0} \;\Leftrightarrow\; \mathbf{D_H = 0} \;\Leftrightarrow\; \mathbf{X_1 = \cdots = X_{n-f}}.$$

[1] The definition of embedding undirected unit-capacity spanning trees in undirected graphs is similar to embedding directed unit-capacity spanning trees in directed graphs (by dropping the direction of edges).

For the following discussion, it is convenient to reorder the elements of $\mathbf{D_H}$ into

$$\tilde{\mathbf{D}}_{\mathbf{H}} = \Big(D_1(1) \ \cdots \ D_{n-f-1}(1) \ \ D_1(2) \ \cdots \ D_{n-f-1}(2) \ \cdots \ D_1(\rho_k) \ \cdots \ D_{n-f-1}(\rho_k)\Big),$$

so that the $(\beta-1)(n-f-1)+1$-th through the $\beta(n-f-1)$ elements of $\tilde{\mathbf{D}}_{\mathbf{H}}$ represent the difference between $\mathbf{X_i}$ $(i = 1, \cdots, n-f-1)$ and $\mathbf{X_{n-f}}$ in the β-th element.

We also reorder the rows of each spanning matrix $\mathbf{S_q}$ $(q = 1, \cdots, \rho_k)$ accordingly to match with the order of elements in $\tilde{D}_H$. After reordering, $\mathbf{S_q}$ becomes $\tilde{\mathbf{S}}_{\mathbf{q}}$ and has the following structure:

$$\tilde{\mathbf{S}}_{\mathbf{q}} = \begin{pmatrix} \mathbf{A_q S_{q,1}} \\ \mathbf{A_q S_{q,2}} \\ \vdots \\ \mathbf{A_q S_{q,\rho_k}} \end{pmatrix}. \tag{B.1}$$

Here $\mathbf{A_q}$ is a $(n-f-1)\times(n-f-1)$ square matrix, and it is called the *adjacency* matrix of the spanning tree corresponding to $\mathbf{S_q}$. $\mathbf{A_q}$ is formed as follows. Suppose that the r-th column of $\mathbf{S_q}$ corresponds to a coded symbol sent over a directed edge (i,j) in H, then

1. If $i \neq n-f$ and $j \neq n-f$, then the r-th column of $\mathbf{A_q}$ has the i-th element as 1 and the j-th element as -1, the remaining entries in that column are all 0;

2. If $i = n-f$, then the j-th element of the r-th column of $\mathbf{A_q}$ is set to -1, the remaining elements of that column are all 0;

3. If $j = n-f$, then the i-th element of the r-th column of $\mathbf{A_q}$ is set to 1, the remaining elements of that column are all 0.

For example, suppose $\overline{H}$ is the graph shown in Figure 3.2(b), and $\mathbf{S_q}$ corresponds to a spanning tree of H consisting of the dotted edges in Figure 3.2(d). Suppose that we index the corresponding directed edges in the graph shown in Figure 3.2(a) in the following order: (2,3), (1,4), (4,3). The result-

ing adjacency matrix $\mathbf{A_q} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & -1 \end{pmatrix}$.

On the other hand, each $\mathbf{S_{q,p}}$ is a $(n-f-1) \times (n-f-1)$ square diagonal matrix. The r-th diagonal element of $\mathbf{S_{q,p}}$ equals to the p-th coefficient used to compute the coded symbol corresponding to the r-th column of $\mathbf{S_q}$. For example, suppose the first column of $\mathbf{S_q}$ corresponds to a coded symbol $3X_1(1) + 4X_1(2)$ being sent on link $(1, 2)$. Then the first diagonal element of $\mathbf{S_{q,1}}$ is 3 and the first diagonal element of $\mathbf{S_{q,2}}$ is 4.

So after the above reordering, $\mathbf{M_H}$ can be written as $\tilde{\mathbf{M}}_\mathbf{H}$ that has the following structure:

$$\tilde{\mathbf{M}}_\mathbf{H} = \begin{pmatrix} \mathbf{A_1 S_{1,1}} & \mathbf{A_2 S_{2,1}} & \cdots & \mathbf{A}_{\rho_k} \mathbf{C}_{\rho_k,1} \\ \mathbf{A_1 S_{1,2}} & \mathbf{A_2 S_{2,2}} & \cdots & \mathbf{A}_{\rho_k} \mathbf{C}_{\rho_k,2} \\ \vdots & & \ddots & \vdots \\ \mathbf{A_1 S_{1,\rho_k}} & \mathbf{A_2 S_{2,\rho_k}} & \cdots & \mathbf{A}_{\rho_k} \mathbf{S}_{\rho_k,\rho_k} \end{pmatrix}. \tag{B.2}$$

Notice that $\tilde{\mathbf{M}}_\mathbf{H}$ is obtained by permuting the rows of $\mathbf{M_H}$. So showing that $\mathbf{M_H}$ is invertible is equivalent to showing that $\tilde{\mathbf{M}}_\mathbf{H}$ is invertible.

Define $\mathbf{M_q} = \begin{pmatrix} \mathbf{A_1 S_{1,1}} & \cdots & \mathbf{A_q S_{q,1}} \\ \vdots & \ddots & \vdots \\ \mathbf{A_1 S_{1,q}} & \cdots & \mathbf{A_q S_{q,q}} \end{pmatrix}$ for $1 \leq q \leq \rho_k$. Note that $\mathbf{M_{q1}}$ is a sub-matrix of $\mathbf{M_{q2}}$ when $q1 < q2$, and $\mathbf{M}_{\rho_k} = \tilde{\mathbf{M}}_\mathbf{H}$. We prove the following lemma:

Lemma B.1 *For any $\rho_k \leq U_k/2$, with probability at least $\left(1 - \frac{n-f-1}{2^{L/\rho_k}}\right)^{\rho_k}$, matrix $\tilde{\mathbf{M}}_\mathbf{H}$ is invertible. Hence $\mathbf{M_H}$ is also invertible with the same probability.*

Proof: We now show that each $\mathbf{M_q}$ is invertible with probability at least $\left(1 - \frac{n-f-1}{2^{L/\rho_k}}\right)^q$ for all $q \leq \rho_k$. The proof is done by induction, with $q = 1$ being the base case.

Base Case: $q = 1$

$$\mathbf{M_1} = \mathbf{A_1 S_{1,1}}. \tag{B.3}$$

As shown later in Appendix B.3.3, $\mathbf{A_q}$ is always invertible and $\det(\mathbf{A_q}) = \pm 1$. Since $\mathbf{S_{1,1}}$ is a $(n-f-1)$-by-$(n-f-1)$ diagonal matrix, it is invertible provided that all its $(n-f-1)$ diagonal elements are non-zero. Remember that the diagonal elements of $\mathbf{S_{1,1}}$ are chosen uniformly and independently from $GF(2^{L/\rho_k})$. The probability that they are all non-zero is $\left(1-\frac{1}{2^{L/\rho_k}}\right)^{n-f-1} \geq 1-\frac{n-f-1}{2^{L/\rho_k}}$.

Induction Step: q to $q+1$ when $1 \leq q < \rho_k$

Recall that M_q is a $(n-f-1)q \times (n-f-1)q$ square matrix. The $(n-f-1)(q+1) \times (n-f-1)(q+1)$square matrix $\mathbf{M_{q+1}}$ can be written as

$$\mathbf{M_{q+1}} = \begin{pmatrix} \mathbf{M_q} & \mathbf{P_q} \\ \mathbf{F_q} & \mathbf{A_{q+1}S_{q+1,q+1}} \end{pmatrix} \tag{B.4}$$

where

$$\mathbf{P_q} = \begin{pmatrix} \mathbf{A_{q+1}S_{q+1,1}} \\ \mathbf{A_{q+1}S_{q+1,1}} \\ \vdots \\ \mathbf{A_{q+1}S_{q+1,q}} \end{pmatrix} \tag{B.5}$$

is an $(n-f-1)q \times (n-f-1)$ matrix, and

$$\mathbf{F_q} = \begin{pmatrix} \mathbf{A_1S_{1,q+1}} & \cdots & \mathbf{A_qS_{q,q+1}} \end{pmatrix} \tag{B.6}$$

is an $(n-f-1) \times (n-f-1)q$ matrix.

Assuming that $\mathbf{M_q}$ is invertible, consider the following matrix $\mathbf{M'_{q+1}}$:

$$\begin{aligned}
\mathbf{M'_{q+1}} &= \begin{pmatrix} \mathbf{I_{(n-f-1)q}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A_{q+1}^{-1}} \end{pmatrix} \mathbf{M_{q+1}} \begin{pmatrix} \mathbf{I_{(n-f-1)q}} & \mathbf{-M_q^{-1}P_q} \\ \mathbf{0} & \mathbf{I_{(n-f-1)}} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I_{(n-f-1)q}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A_{q+1}^{-1}} \end{pmatrix} \begin{pmatrix} \mathbf{M_q} & \mathbf{P_q} \\ \mathbf{F_q} & \mathbf{A_{q+1}S_{q+1,q+1}} \end{pmatrix} \begin{pmatrix} \mathbf{I_{(n-f-1)q}} & \mathbf{-M_q^{-1}P_q} \\ \mathbf{0} & \mathbf{I_{(n-f-1)}} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{M_q} & \mathbf{P_q} \\ \mathbf{A_{q+1}^{-1}F_q} & \mathbf{S_{q+1,q+1}} \end{pmatrix} \begin{pmatrix} \mathbf{I_{(n-f-1)q}} & \mathbf{-M_q^{-1}P_q} \\ \mathbf{0} & \mathbf{I_{(n-f-1)}} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{M_q} & \mathbf{0} \\ \mathbf{A_{q+1}^{-1}F_q} & \mathbf{S_{q+1,q+1} - A_{q+1}^{-1}F_qM_q^{-1}P_q} \end{pmatrix}.
\end{aligned}$$

Here $\mathbf{I_{(n-f-1)q}}$ and $\mathbf{I_{(n-f-1)}}$, respectively, denote $(n-f-1)q \times (n-f-1)q$

and a $(n-f-1)\times(n-f-1)$ identity matrices. For this proof, we need to show that $\mathbf{M_{q+1}}$ is invertible, i.e., $\det(\mathbf{M_{q+1}}) \neq 0$. Note that $|\det(\mathbf{M'_{q+1}})| = |\det(\mathbf{M_{q+1}})|$, since the matrix multiplied at the left has determinant ± 1, and the matrix multiplied at the right has determinant 1. So it suffices to show that $\mathbf{M'_{q+1}}$ is invertible.

Observe that the diagonal elements of the $(n-f-1)\times(n-f-1)$ diagonal matrix $\mathbf{S_{q+1,q+1}}$ are chosen independently from $\mathbf{A_{q+1}^{-1}F_qM_q^{-1}P_q}$. Then as shown later in Appendix B.3.4, $\mathbf{S_{q+1,q+1}} - \mathbf{A_{q+1}^{-1}F_qM_q^{-1}P_q}$ is invertible with probability at least $1-\frac{n-f-1}{2^{L/\rho_k}}$, given that $\mathbf{M_q}$ is invertible, which happens with probability at least $\left(1-\frac{n-f-1}{2^{L/\rho_k}}\right)^q$ according to the induction assumption. So we have

$$\begin{aligned}
\Pr\{\mathbf{M_{q+1}} \text{ is invertible}\} &= \Pr\{\mathbf{M'_{q+1}} \text{ is invertible}\} \\
&\geq \left(1-\frac{n-f-1}{2^{L/\rho_k}}\right)^q \left(1-\frac{n-f-1}{2^{L/\rho_k}}\right) \quad \text{(B.7)} \\
&= \left(1-\frac{n-f-1}{2^{L/\rho_k}}\right)^{q+1}. \quad \text{(B.8)}
\end{aligned}$$

This completes the induction. Now we can see that $\mathbf{M_{\rho_k}} = \tilde{\mathbf{M}}_{\mathbf{H}}$ is invertible with probability

$$\begin{aligned}
&\geq \left(1-\frac{n-f-1}{2^{L/\rho_k}}\right)^{\rho_k} \quad \text{(B.9)} \\
&\geq 1-\frac{(n-f-1)\rho_k}{2^{L/\rho_k}} \quad \text{(B.10)} \\
&\to 1, \text{ as } L\to\infty. \quad \text{(B.11)}
\end{aligned}$$

□

Now we have proved that there exists a set of coding matrices $\{\mathbf{C_e}|e\in\mathcal{E}_k\}$ such that the resulting $\mathbf{C_H}$ satisfies the condition that $\mathbf{D_HC_H} = \mathbf{0}$ if and only if $\mathbf{D_H} = \mathbf{0}$.

B.3.2 For All Subgraphs in Ω_k

In this section, we are going to show that, for $\mathcal{G}_k$, if the coding matrices $\{\mathbf{C_e}|e\in\mathcal{E}_k\}$ are generated as described in Theorem 3.1, then with non-zero probability the set of square matrices $\{\mathbf{M_H}|H\in\Omega_k\}$ are all invertible

simultaneously. When this is true, there exists a set of coding matrices that is correct.

To show that $\mathbf{M_H}$'s for all $H \in \Omega_k$ are simultaneously invertible with non-zero probability, we consider the product of all these square matrices:

$$\prod_{H \in \Omega_k} \mathbf{M_H}.$$

According to Lemma B.1, each $\mathbf{M_H}$ ($H \in \Omega_k$) is invertible with non-zero probability. It implies that $\det(\mathbf{M_H})$ is a non-identically-zero polynomial of the random coding coefficients of degree at most $(n-f-1)\rho_k$ (recall that $\mathbf{M_H}$ is a square matrix of size $(n-f-1)\rho_k$.). So

$$\det\left(\prod_{H \in \Omega_k} \mathbf{M_H}\right) = \prod_{H \in \Omega_k} \det\left(\mathbf{M_H}\right)$$

is a non-identically-zero polynomial of the random coefficients of degree at most $|\Omega_k|(n-f-1)\rho_k$. Notice that each coded symbol is used once in each subgraph H. So each random coefficient appears in at most one column in each $\mathbf{M_H}$. It follows that the largest exponent of any random coefficient in $\det\left(\prod_{H \in \Omega_k} \mathbf{M_H}\right)$ is at most $|\Omega_k|$.

According to Lemma 1 of [77], the probability that $\det\left(\prod_{H \in \Omega_k} \mathbf{M_H}\right)$ is non-zero is at least

$$\left(1 - 2^{-L/\rho_k}|\Omega_k|\right)^{(n-f-1)\rho_k} \geq 1 - 2^{-L/\rho_k}\left[|\Omega_k|(n-f-1)\rho_k\right].$$

According to the way $\mathcal{G}_k$ is constructed and the definition of Ω_k, it should not be hard to see that $\mathcal{G}_k$ is a subgraph of $\mathcal{G}_1 = \mathcal{G}$, and $\Omega_k \subseteq \Omega_1$. Notice that $|\Omega_1| = \binom{n}{n-f}$. So $|\Omega_k| \leq \binom{n}{n-f}$ and Theorem 3.1 follows.

B.3.3 Proof that $\mathbf{A_q}$ is Invertible

Given an adjacency matrix $\mathbf{A_q}$, let us call the corresponding spanning tree of $\overline{H}$ as T_q. For edges in T_q incident on node $n-f$, the corresponding columns in $\mathbf{A_q}$ have exactly one non-zero entry. Also, the column corresponding to an edge that is incident on node i has a non-zero entry in row i. Since there must be at least one edge in T_q that is incident on node $n-f$, there must be at

least one column of $\mathbf{A_q}$ that has only one non-zero element. Also, since every node is incident on at least one edge in T_q, every row of $\mathbf{A_q}$ has at least one non-zero element(s). Since there is at most one edge between every pair of nodes in T_q, no two columns in $\mathbf{A_q}$ are non-zero in identical rows. Therefore, by column manipulation, we can transform matrix $\mathbf{A_q}$ into another matrix in which every row and every column has exactly one non-zero element. Hence $\det(\mathbf{A_q})$ equals to either 1 or -1, and $\mathbf{A_q}$ is invertible.

B.3.4 Proof that $\mathbf{S_{q+1,q+1} - A_{q+1}^{-1} F_q M_q^{-1} P_q}$ is Invertible

Consider $\mathbf{W}$ to be an arbitrary *fixed* $w \times w$ matrix. Consider a random $w \times w$ diagonal matrix $\mathbf{S}$ with w diagonal elements $s_1, \cdots, s_w$.

$$\mathbf{S} = \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & s_w \end{pmatrix}. \tag{B.12}$$

The diagonal elements of $\mathbf{S}$ are selected independently and uniformly randomly from $GF(2^{\rho_k})$. Then we have:

Lemma B.2 *The probability that the $w \times w$ matrix $\mathbf{S} - \mathbf{W}$ is invertible is lower-bounded by:*

$$\Pr\{(\mathbf{S} - \mathbf{W}) \textit{ is invertible}\} \geq 1 - \frac{w}{2^{\rho_k}}. \tag{B.13}$$

Proof: Consider the determinant of matrix $\mathbf{S} - \mathbf{W}$.

$$\begin{aligned} \det(\mathbf{S} - \mathbf{W}) &= \det \begin{pmatrix} (s_1 - W_{1,1}) & -R_{1,2} & \cdots & -W_{1,w} \\ -R_{2,1} & (s_2 - W_{2,2}) & \cdots & -R_{2,w} \\ \vdots & & \ddots & \vdots \\ -R_{w,1} & \cdots & -W_{w,w-1} & (s_r - R_{w,w}) \end{pmatrix} \\ &= (s_1 - W_{1,1})(s_2 - W_{2,2}) \cdots (s_w - W_{w,w}) \quad + \quad \text{other terms} \\ &= \Pi_{i=1}^{w} s_i \quad + \quad W_{w-1} \end{aligned} \tag{B.14}$$

The first term above, $\Pi_{i=1}^{w} s_i$, is a degree-w polynomial of $s_1, \cdots, s_w$. W_{w-1} is a polynomial of degree at most $w-1$ of $s_1, \cdots, s_w$, and it represents the

remaining terms in $\det(\mathbf{S}-\mathbf{W})$. Notice that $\det(\mathbf{S}-\mathbf{W})$ cannot be identically zero since it contains only one degree-w term. Then by the Schwartz-Zippel Theorem, the probability that $\det(\mathbf{S}-\mathbf{R})=\mathbf{0}$ is $\leq w/2^{\rho_k}$. Since $\mathbf{S}-\mathbf{W}$ is invertible if and only if $\det(\mathbf{S}-\mathbf{W})\neq\mathbf{0}$, we conclude that

$$\Pr\{(\mathbf{S}-\mathbf{W}) \text{ is invertible}\} \geq 1-\frac{w}{2^{\rho_k}}. \tag{B.15}$$

By setting $\mathbf{S}=\mathbf{S_{q+1,q+1}}$, $\mathbf{W}=\mathbf{A_{q+1}^{-1}F_qM_q^{-1}P_q}$, and $w=n-f-1$, we prove that $\mathbf{S_{q+1,q+1}}-\mathbf{A_{q+1}^{-1}F_qM_q^{-1}P_q}$ is invertible with probability at least $1-\frac{n-f-1}{2^{L/\rho_k}}$.

□

B.4 Throughput of NAB

First consider the time cost of each operation in instance k of NAB :

- **Phase 1**: It takes $L/\gamma_k \leq L/\gamma^*$ time units, since unreliable broadcast from the source node 1 at rate γ_k is achievable and $\gamma_k \geq \gamma^*$, as discussed in Appendix B.1.

- **Phase 2 – Equality check**: As discussed previously, it takes $L/\rho_k \leq L/\rho^*$ time units.

- **Phase 2 – Broadcasting outcomes of equality check**: To reliably broadcast the 1-bit flags from the equality check algorithm, a previously proposed Byzantine broadcast algorithm, such as [35], is used. The algorithm from [35], denoted as `Broadcast_Binary` hereafter, reliably broadcasts 1 bit by communicating no more than $P(n)$ bits in a *complete* graph, where $P(n)$ is a polynomial of n. In our setting, $\mathcal{G}$ might not be complete. However, the connectivity of $\mathcal{G}$ is at least $2f+1$. It is well-known that, in a graph with connectivity at least $2f+1$ and at most f faulty nodes, reliable *end-to-end* communication from any node i to any other node j can be achieved by sending the same copy of data along a set of $2f+1$ node-disjoint paths from node i to node j and taking the majority at node j. By doing this, we can emulate a complete graph in an incomplete graph $\mathcal{G}$. Since every simple path in $\mathcal{G}$ is at most $n-1$ hops long, and each edge in $\mathcal{E}$ has capacity at

least 1 bit/time unit, it takes at most $n-1$ time units to emulate in $\mathcal{G}$ the transmission of 1 bit in the complete graph. Then we can conclude that, by running `Broadcast_Binary` on top of the emulated complete graph, reliably broadcasting the 1-bit flags can be completed in $O(n^\alpha)$ time units, for some constant $\alpha > 0$.

- **Phase 3**: If Phase 3 is performed in instance k, every node i in $\mathcal{V}_k$ uses `Broadcast_Binary` to reliably broadcast all the messages that it claims to have received from other nodes, and sent to the other nodes, during Phase 1 and 2 of the k-th instance. Denote the total capacity of edges in $\mathcal{E}$ as m. Since Phases 1 and 2 takes at most $L/\gamma^* + L/\rho^*$ time units, at most $mL(1/\gamma^* + 1/\rho^*)$ bits have been communicated in $\mathcal{G}$ during Phases 1 and 2. Similar to the discussion above about broadcasting the outcomes of equality check, the time it takes to complete Phase 3 is $O(mL(1/\gamma^* + 1/\rho^*)n^\alpha)$.

Now consider a sequence of $Q > 0$ instances of NAB. As discussed previously, Phase 3 will be performed at most $f(f+1)$ times throughout the execution of the algorithm. So we have the following upper bound of the execution time of Q instances of NAB:

$$t(\mathcal{G}, L, Q, NAB) \leq Q\left(\frac{L}{\gamma^*} + \frac{L}{\rho^*} + O(n^\alpha)\right) + f(f+1)O\left(mL(\frac{1}{\gamma^*} + \frac{1}{\rho^*})n^\alpha\right).$$

Then the throughput of NAB can be lower-bounded by

$$\begin{aligned} T(\mathcal{G}, L, NAB) &= \lim_{Q\to\infty} \frac{LQ}{t(\mathcal{G}, L, Q, NAB)} \\ &\geq \lim_{Q\to\infty} \frac{LQ}{Q\left(\frac{L}{\gamma^*} + \frac{L}{\rho^*} + O(n^\alpha)\right) + f(f+1)O\left(mL(\frac{1}{\gamma^*} + \frac{1}{\rho^*})n^\alpha\right)} \\ &\geq \lim_{Q\to\infty} \left(\frac{\gamma^* + \rho^*}{\gamma^*\rho^*} + O\left(\frac{n^\alpha}{L}\right) + O\left(\frac{m(\gamma^* + \rho^*)n^{\alpha+2}}{\gamma^*\rho^* Q}\right)\right)^{-1} \end{aligned}$$

The last inequality is due to $f < n/3$.

Notice that for a given graph $\mathcal{G}$, $\{n, \gamma^*, \rho^*, \alpha, m\}$ are all constants independent of L and Q. So for sufficiently large values of L and Q, specifically when $L = \Omega(n^\alpha)$ and $Q = \Omega\left(\frac{m(\gamma^*+\rho^*)n^{\alpha+2}}{\gamma^*\rho^*}\right)$, the last two terms in the last inequality becomes negligible compared to the first term, and the throughput of NAB approaches to a value that is at least as large as T_{NAB}, which is

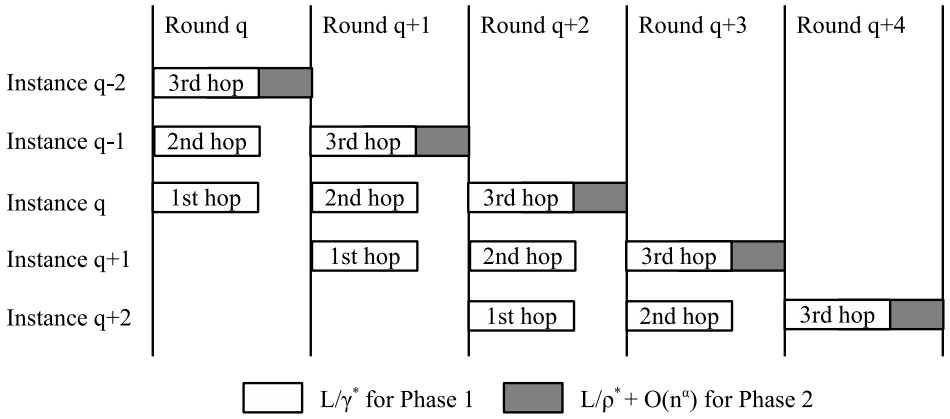


Figure B.1: Example of pipelining

defined

$$T_{NAB}(\mathcal{G}) = \frac{\gamma^*\rho^*}{\gamma^* + \rho^*}. \tag{B.16}$$

In the above discussion, we implicitly assumed that transmissions during the unreliable broadcast in Phase 1 accomplish all at the same time, by assuming no propagation delay. However, when propagation delay is considered, a node cannot forward a message/symbol until it finishes receiving it. So for the k-th instance of NAB, the information broadcast by the source propagates only one hop every L/γ_k time units. So for a large network, the "time span" of Phase 1 can be much larger than L/γ_k. This problem can be solved by pipelining: We divide the time horizon into rounds of $\left(\frac{L}{\gamma^*} + \frac{L}{\rho^*} + O(n^\alpha)\right)$ time units. For each instance of NAB, the L-bit input from the source node 1 propagates one hop per round, using the first L/γ^* time units, until Phase 1 completes. Then the remaining $\left(\frac{L}{\rho^*} + O(n^\alpha)\right)$ time units of the last round is used to perform Phase 2. An example in which the broadcast in Phase 1 takes 3 hops is shown in Figure B.1. In a particular round q, first hop transmissions of the unreliable broadcast in Phase 1 of the q-th instance, the second hop transmissions of the Phase 1 of the $(q-1)$-th instance , and the third hop transmissions of the phase 1 of the $(q-2)$-th instance are performed simultaneously during the first L/γ^* time units. Phase 1 of the $(q-2)$-th instance completes by then. Then failure detection of Phase 2 of the $(q-2)$-th instance is performed using the remaining $L/\rho^* + O(n^\alpha)$ unit time, illustrated as the gray area. During the gray area, all links in the network are used.

By pipelining, we achieve the lower bound from Equation 3.6.

B.5 Construction of Γ

A subgraph of $\mathcal{G}$ belonging to Γ is obtained as follows: We will say that edges in $W \subset \mathcal{E}$ are "explainable" if there exists a set $F \subset \mathcal{V}$ such that (i) F contains at most f nodes, and (ii) each edge in W is incident on at least one node in F. Set F is then said to "explain set W".

Consider each *explainable* set of edges $W \subset \mathcal{E}$. Suppose that $F_1, \cdots, F_\Delta$ are all the subsets of $\mathcal{V}$ of size $\leq f$ that *explain* edge set W. A subgraph Ψ_W of $\mathcal{G}$ is obtained by removing edges in W from $\mathcal{E}$, and nodes in $\bigcap_{\delta=1}^{\Delta} F_\delta$ from $\mathcal{V}$.[2] In general, Ψ_W above may or may not contain the source node 1. Only those Ψ_W's that do contain node 1 belong to Γ.

B.6 Proof of Theorem 3.2

In arbitrary point-to-point network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the capacity of the BB problem with node 1 being the source and up to $f < n/3$ faults satisfies the upper bounds proved in Sections B.6.1 and B.6.2.

B.6.1 $C_{BB}(\mathcal{G}) \leq \gamma^*$

Proof: Consider any $\Psi_W \in \Gamma$ and let W be the set of edges in $\mathcal{G}$ but not in Ψ_W. By the construction of Γ, there must be at least one set $F \subset \mathcal{V}$ that explains W and does not contain the source node 1. We are going to show that $C_{BB}(\mathcal{G}) \leq MINCUT(\Psi_W, 1, i)$ for every node $i \neq 1$ that is in Ψ_W.

Notice that there must exist a set of nodes that explains W and does not contain node 1; otherwise node 1 is not in Ψ_W. Without loss of generality, assume that F_1 is one such set of nodes. So F_1 does not contain source node 1.

First consider any node $i \neq 1$ in Ψ_W such that $i \notin F_1$. Let all the nodes in F_1 be faulty such that they refuse to communicate over edges in W, but otherwise behave correctly. In this case, since the source is fault-free, node i must be able to receive the L-bit input that node 1 is trying to broadcast. So $C_{BB}(\mathcal{G}) \leq MINCUT(\Psi_W, 1, i)$.

[2] It is possible that Ψ_W for different W may be identical. This does not affect the correctness of our algorithm.

Next we consider a node $i \neq 1$ in Ψ_W such that $i \in F_1$, if it exists. Notice that node i cannot be contained in all sets of nodes that explain W; otherwise, node i is not in Ψ_W. Then there are only two possibilities:

1. There exists another set F that explains W and contains neither node 1 nor node i. In this case, $C_{BB}(\mathcal{G}) \leq MINCUT(\Psi_W, 1, i)$ according to the above argument by replacing F_1 with F.

2. Otherwise, there must exist a set $F_2 \neq F_1$ that explains W such that $i \notin F_2$ and $1 \in F_2$.

   Define $V^- = \mathcal{V} - F_1 - F_2$. V^- is not empty since F_1 and F_2 both contain at most f nodes and there are $n \geq 3f + 1$ nodes in $\mathcal{V}$. Consider two scenarios with the same input value x: (1) Nodes in F_1 (which does not contain node 1) are faulty and refuse to communicate over edges in W, but otherwise behave correctly; and (2) Nodes in F_2 (which contains node 1) are faulty and refuse to communicate over edges in W, but otherwise behave correctly. In both cases, nodes in V^- are fault-free.

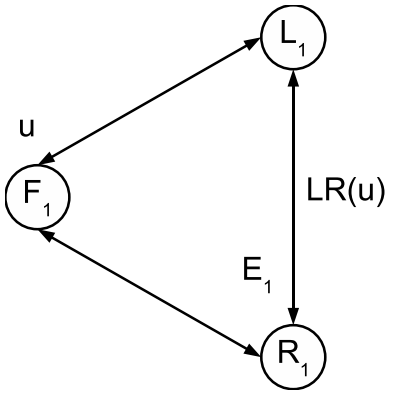
   Observe that among edges connecting nodes in V^- to/from nodes in $F_1 \cup F_2$, only edges connecting nodes in V^- to/from nodes $F_1 \cap F_2$ could have been removed, because otherwise W cannot be explained by both F_1 and F_2. So nodes in V^- cannot distinguish between the two scenarios above. In scenario (1), the source node 1 is not faulty. Hence nodes in V^- must agree with the value x that node 1 is trying to broadcast, according to the validity condition. Since nodes in V^- cannot distinguish between the two scenarios, they must also set their outputs to x in scenario (2), even though in this case the source node 1 is faulty. Then according to the agreement condition, node i must agree with nodes in V^- in scenario (2), which means that node i also has to learn x. So $C_{BB}(\mathcal{G}) \leq MINCUT(\Psi_W, 1, i)$.

This completes the proof. □

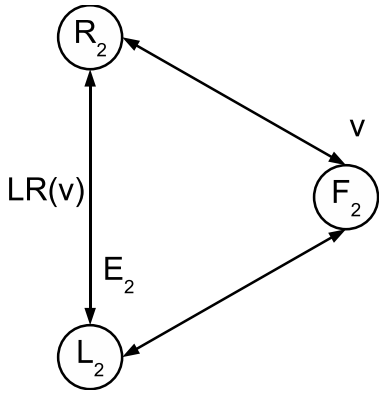
B.6.2 $C_{BB}(\mathcal{G}) \leq 2\rho^*$

Proof: For a subgraph $H \in \Omega_1$ (and accordingly $\overline{H} \in \overline{\Omega}_1$), denote

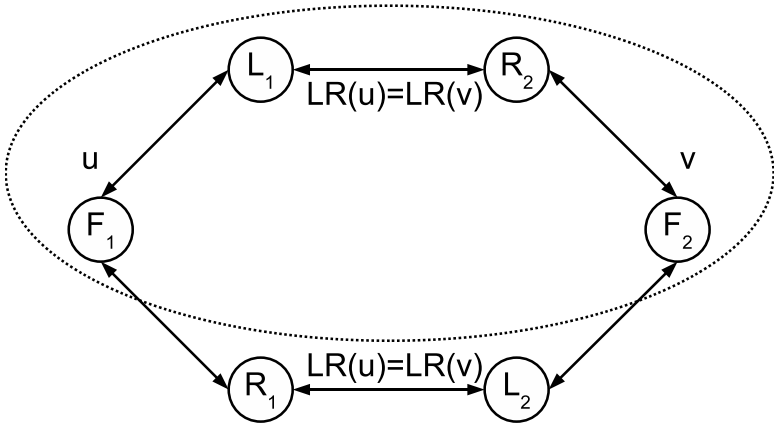
$$U_H = \min_{nodes\ i,j\ in\ H} MINCUT(\overline{H}, i, j).$$



(a) Scenario 1. Source is in F_1, and the input value is u.



(b) Scenario 2. Source is in F_2, and input value is v.



(c) Scenario 3

Figure B.2: Three scenarios for the proof of $C_{BB}(\mathcal{G}) \leq 2\rho^*$.

We will prove the upper bound by showing that $C_{BB}(G) \leq U_H$ for every $H \in \Omega_1$.

Suppose on the contrary that Byzantine broadcast can be done at a rate $R > U_H + \epsilon$ for some constant $\epsilon > 0$. So there exists a BB algorithm, named $\mathcal{A}$, that can broadcast $t(U_H + \epsilon)$ bits in using t time units, for some $t > 0$.

Let E be a set of edges in H that corresponds to one of the minimum-cuts in $\overline{H}$. In other words, $\sum_{e \in E} z_e = U_H$, and the nodes in H can be partitioned into two non-empty sets $\mathcal{L}$ and $\mathcal{R}$ such that $\mathcal{L}$ and $\mathcal{R}$ are disconnected from each other if edges in E are removed. Also denote F as the set of nodes that are in $\mathcal{G}$ but not in H. Since H contains $(n - f)$ nodes, F contains f nodes.

Notice that in t time units, at most $tU_H < t(U_H + \epsilon)$ bits of information can be sent over edges in E. According to the pigeonhole principle, there must exist two different input values of $t(U_H + \epsilon)$ bits, denoted as u and v,

such that in the absence of misbehavior, broadcasting u and v with algorithm $\mathcal{A}$ results in the same communication pattern over edges in E.

There are two possible cases: (1) F contains the source node 1; and (2) F does not contain the source node 1.

First case: F contains the source node 1 Consider the three scenarios using algorithm $\mathcal{A}$:

1. Node 1 broadcasts u, and none of the nodes misbehaves. So all nodes should set their outputs to u.

2. Node 1 broadcasts v, and none of the nodes misbehaves. So all nodes should set their outputs to v.

3. Nodes in F are faulty (includes the source node 1). The faulty nodes in F behave to nodes in $\mathcal{L}$ as in scenario 1, and behave to nodes in $\mathcal{R}$ as in scenario 2.

Next, we will show that nodes in $\mathcal{L}$ cannot distinguish scenario 1 from scenario 3, and nodes in $\mathcal{R}$ cannot distinguish scenario 2 from scenario 3. For this purpose, we construct 3 state machines, one for each of the above scenarios in Figure B.2. The state machine for scenario 1 is shown in Figure B.2(a): it is a copy of $\mathcal{G}$, and the circles F_1, L_1 and R_1, respectively, represent copies of the nodes in F, $\mathcal{L}$ and $\mathcal{R}$. The bidirectional edge E_1 between L_1 and R_1 represents copies of the edges connecting nodes in $\mathcal{L}$ to/from nodes in $\mathcal{R}$, i.e., edges in E. Similarly, the other two bidirectional edges represent the edges connecting nodes in F to/from nodes in $\mathcal{L}$, and the edges connecting nodes in F to/from nodes in $\mathcal{R}$, respectively. Edges connecting nodes within each set are not shown in the figure. The copy of source node 1 in F_1 is given input value u. The information being communicated between L_1 and R_1 over edges in E_1 is denoted as $LR(u)$. Similarly, Figure B.2(b) is the state machine for scenario 2, with the input value to the copy of the source node being v, and the corresponding information communicated between L_2 and R_2 over edges in E_2 is denoted as $LR(v)$.

The state machine in Figure B.2(c) is constructed as follows: (1) detach E_1 from R_1 and attach it to R_2 as $\mathcal{L}$ and $\mathcal{R}$ are connected by E; (2) similarly, detach E_2 from R_2 and attach it to R_1 as $\mathcal{L}$ and $\mathcal{R}$ are connected by E. Recall that $LR(u) = LR(v)$. It follows that the behaviors of the nodes in

$F_1, L_1, R_1, F_2, L_2, R_2$ are not affected by above modification of switching edges in E_1 and E_2. In particular, L_1 behave identically in both state machines from Figure B.2(a) and Figure B.2(c); and R_2 behave identically in both state machines from Figure B.2(b) and Figure B.2(c). For scenario 3, we let $\mathcal{L} = L_1$, $\mathcal{R} = R_2$, and the faulty nodes in F behave to $\mathcal{L}$ as F_1 and behave to $\mathcal{R}$ as F_2 in Figure B.2(c).

Now we have showed that nodes in $\mathcal{L}$ cannot distinguish scenario 1 from scenario 3, and nodes in $\mathcal{R}$ cannot distinguish scenario 2 from scenario 3. So in scenario 3, nodes in $\mathcal{L}$ set their outputs to u and nodes in $\mathcal{R}$ set their outputs to v. This violates the agreement condition and contradicts with the assumption that $\mathcal{A}$ solves BB at rate $U_H + \epsilon$. Hence $C_{BB}(\mathcal{G}) \leq U_H$.

Second case: F does not contain the source node 1 Without loss of generality, suppose that node 1 is in $\mathcal{L}$. Consider the following three scenarios:

1. Node 1 broadcasts u, and none of the nodes misbehave. So all nodes should set their outputs to u.

2. Node 1 broadcasts v, and none of the nodes misbehave. So all nodes should set their outputs to v.

3. Node 1 broadcasts u, and nodes in F are faulty. The faulty nodes in F behave to nodes in $\mathcal{L}$ as in scenario 1, and behave to nodes in $\mathcal{R}$ as in scenario 2.

For this part, we construct 3 state machines, one for each of the above scenarios, that are almost the same as those in Figure B.2 for the first case, with the modification that now the source node is in L_1 and L_2 in Figures B.2(a) and B.2(b), respectively. Similar to the first case, nodes in $\mathcal{L}$ cannot distinguish scenario 1 from scenario 3, and nodes in $\mathcal{R}$ cannot distinguish scenario 2 from scenario 3. So in scenario 3, nodes in $\mathcal{L}$ set their outputs to u and nodes in $\mathcal{R}$ set their outputs to v. This violates the agreement condition and contradicts with the assumption that $\mathcal{A}$ solves BB at rate $U_H + \epsilon$. Hence $C_{BB}(\mathcal{G}) \leq U_H$, and this completes the proof. □

B.7 Correctness of Algorithm 3.7.6

As in network coding [46], for sufficiently large value of L and R no greater than the upper bound from Corollary 3.1, as discussed later, there exist sets of coefficients that make every subset of R coded symbols computed in Algorithm 3.7.6 correspond to a set of R *linearly independent* linear combinations of the R data symbols of $\mathbf{X}$. We assume in the following discussion that one set of such coefficients is chosen. A set of coded symbols are said to be linearly independent if the corresponding linear combinations of the data symbols are linearly independent. From basic linear algebra, we know that there exists a unique vector $\mathbf{Y}$ that satisfies all the linear combinations represented by any subset of R linearly independent coded symbols. We call vector $\mathbf{Y}$ the unique solution to the set of R linearly independent coded symbols.

Notice that the source node 1 can be in dispute with at most one peer, otherwise it must have been identified as faulty and the fault-free peers should have terminated the algorithm with a default output. So there are three cases: the source is fault-free, the source is faulty and not in dispute with any peer, and the source is faulty and in dispute with one peer. We consider these three cases separately.

For the following discussion, denote $z_{i,j}$ as the capacity of edge $e = (i, j)$. With this notation, Corollary 3.1 can be expressed as follows:

$$\begin{aligned} z_{j,i} + z_{k,i} &\geq C_{BB4}, \quad i \in \{2,3,4\}; j,k \in \{1,2,3,4\};\ i \neq j, i \neq k, j \neq k; \\ z_{1,i} + z_{1,j} &\geq C_{BB4}, \quad i \neq 1, j \neq 1, i \neq j. \end{aligned}$$

To prove the correctness of Algorithm 3.7.6, consider any R that is no greater than the upper bound from Corollary 3.1, i.e.,

$$z_{j,i} + z_{k,i} \geq R, \quad i = 2,3,4;\ i \neq j, i \neq k, j \neq; \tag{B.17}$$

$$z_{1,i} + z_{1,j} \geq R, \quad i \neq 1, j \neq 1, i \neq j. \tag{B.18}$$

Source is fault-free: Consider any two fault-free peer nodes i and j. According to the manner in which coded symbols are transmitted in Algorithm

3.7.6, it is easy to see that node i receives at least

$$\begin{aligned} z_{1,i} + \min(z_{1,j}, z_{j,i}) &= \min(z_{1,i} + z_{1,j}, z_{1,i} + z_{j,i}) \\ &\geq R \end{aligned}$$

untampered linearly independent coded symbols, either directly from source or via the other fault-free peer node j. Since $\mathbf{X}$ is the unique solution to any set of untampered linearly independent coded symbols, either node i finds the unique solution to the coded symbols it has received as $\mathbf{Y_i} = \mathbf{X}$, or there does not exist a unique solution, in which case failure is detected. So when the source is fault-free, either all fault-free peers decide on the correct output $\mathbf{X}$, or at least one node detects a failure.

Source is faulty and not in dispute with any peer: In this case, all peer nodes are fault-free; they are never in dispute with each other. According to Equation B.17, we have

$$\begin{aligned} & (z_{2,3} + z_{3,2}) + (z_{2,4} + z_{4,2}) + (z_{3,4} + z_{4,3}) \\ = \; & (z_{3,2} + z_{4,2}) + (z_{2,3} + z_{4,3}) + (z_{2,4} + z_{3,4}) \geq 3R. \end{aligned}$$

This implies at least one of the terms $(z_{2,3} + z_{3,2})$, $(z_{2,4} + z_{4,2})$ and $(z_{3,4} + z_{4,3})$ must exceed R. Without loss of generality, suppose that $z_{2,3} + z_{3,2} \geq R$.

Now, let us consider the number of coded symbols nodes 2 and 3 exchange in step 2 on links (2,3) and (3,2) together. Observe that node 2 sends $min\{z_{1,2}, z_{2,3}\}$ coded symbols to node 3 on (2,3), and node 3 sends $min\{z_{1,3}, z_{3,2}\}$ to node 2 on link (3,2). So the number of coded symbols they exchange on links (2,3) and (3,2) together is

$$\begin{aligned} & \min\{z_{1,2}, z_{2,3}\} + min\{z_{1,3}, z_{3,2}\} && \text{(B.19)} \\ = \; & \min\{z_{1,2} + z_{1,3}, z_{1,2} + z_{3,2}, z_{2,3} + z_{1,3}, z_{2,3} + z_{3,2}\} && \text{(B.20)} \\ \geq \; & min\{R, z_{2,3} + z_{3,2}\} && \text{(B.21)} \end{aligned}$$

The reason for the $\geq$ in Equation B.21 is that each of the first three terms on the right-hand side of Equation B.20, namely $z_{1,2} + z_{1,3}$, $z_{1,2} + z_{3,2}$, $z_{2,3} + z_{1,3}$, is $\geq R$, as per Equations B.17 and B.18. $z_{2,3} + z_{1,3} \geq R$ and Equation B.21 together imply that after step 2, nodes 2 and 3 share at least R coded symbols.

That is, among the symbols nodes 2 and 3 have received, there are R identical symbols. Thus, nodes 2 and 3 will not agree on different data symbols (since the agreed data must satisfy linear combinations corresponding to all received coded symbols).

Thus, either at least one of nodes 2 and 3 will detect misbehavior by node 1, or all the symbols they have received will be consistent with an identical vector $\mathbf{Y}$ (of R symbols). In the former case, the misbehavior by node 1 is detected. In the latter case, neither node 2 nor 3 detects the misbehavior. In this case, we will now consider what happens at node 4. In particular, there are three possibilities:

- $z_{2,4} + z_{4,2} \geq R$: Similar to the above argument for nodes 2 and 3, we can argue that nodes 2 and 4 will have at least R coded symbols in common, and therefore, they will not agree on two different data vectors. This, combined with the fact that nodes 2 and 3 will also not agree on two different vectors, implies that if none of the three fault-free peers detects a misbehavior, then they will agree on an identical vector $\mathbf{Y}$.

- $z_{3,4} + z_{4,3} \geq R$: This case is similar to the previous case.

- $z_{2,4}+z_{4,2} < R$ and $z_{3,4}+z_{4,3} < R$: In this case, similar to Equation B.21, we can show that:

$$\min\{z_{1,2}, z_{2,4}\} + min\{z_{1,4}, z_{4,2}\} \geq \min\{R, z_{2,4} + z_{4,2}\}$$

$$\min\{z_{1,3}, z_{3,4}\} + min\{z_{1,4}, z_{4,3}\} \geq \min\{R, z_{3,4} + z_{4,3}\}.$$

  This implies that these four links are all "saturated" (that is, the number of coded symbols sent on each of these links in round 2 is equal to the link capacity). It follows that node 4 receives $z_{2,4} + z_{3,4} \geq R$ coded symbols from nodes 2 and 3 together, and node 4 has at least R coded symbols in common with the union of symbols available to nodes 2 and 3. Since nodes 2 and 3 have not detected a misbehavior, these R symbols must all be consistent with the solution obtained at nodes 2 and 3 both. Thus, node 4 cannot possibly decide on an output that is different from that agreed upon by nodes 2 and 3. Thus, it follows that

either at least one of the peers will detect the misbehavior by node 1, or they will all agree.

Source is faulty and in dispute with one peer: Without loss of generality, assume that the faulty source node 1 is in dispute with the fault-free peer node 4. By the end of step 2, node 2 has received $z_{1,2}+\min(z_{1,3}, z_{3,2}) \geq R$ coded symbols from source node 1 and node 3 together. Similarly, node 3 has received $z_{1,3} + \min(z_{1,2}, z_{2,3}) \geq R$ coded symbols. So the coded symbols nodes 2 and 3 compute and send to node 4 in step 3 satisfy the linearly independent requirement. Then in steps 2-3, node 4 receives $z_{2,4} + z_{3,4} \geq R$ coded symbol from nodes 2 and 3 together. As before, the coded symbols node 4 computes and sends in step 3 satisfy the linearly independent requirement.

Then, in step 3, node 4 sends $\min(z_{4,2}, z_{4,3})$ *identical* coded symbols to both nodes 2 and 3. It then follows that nodes 2 and 3 share

$$\begin{aligned} & \min(z_{1,2}, z_{2,3}) + \min(z_{1,3}, z_{3,2}) + \min(z_{4,2}, z_{4,3}) \\ = \; & \min(z_{1,2} + z_{1,3} + z_{4,2}, z_{1,2} + z_{1,3} + z_{4,3}, \\ & \quad z_{1,2} + z_{3,2} + z_{4,2}, z_{1,2} + z_{3,2} + z_{4,3}, \\ & \quad z_{2,3} + z_{1,3} + z_{4,2}, z_{2,3} + z_{1,3} + z_{4,3}, \\ & \quad z_{2,3} + z_{3,2} + z_{4,2}, z_{2,3} + z_{3,2} + z_{4,3}) \end{aligned}$$

linearly independent coded symbols identically. Following directly from Equation B.17 and Equation B.18, all 8 summations in the min function above are $\geq R$. It follows that nodes 2 and 3 have at least R coded symbols in common, and hence they cannot possibly agree on different outputs. Similar to the previous case, it follows that either at least one of the peers will detect the misbehavior by node 1, or they will all agree. This completes the proof.

APPENDIX C

MULTIPARTY EQUALITY FUNCTION COMPUTATION

C.1 Edge Coloring Representation of MEQ-AD(3,K)

From Sections 4.5 and 4.6, we have shown that it is sufficient to study 3-node systems where information is transmitted only on links AB, AC and BC. Let us denote $|s_{AB}|$, $|s_{AC}|$ and $|s_{BC}|$ as the number of different symbols being transmitted on links AB, AC and BC, respectively. Now consider the following simple bipartite graph $G(U, V, E)$, where U and V are the two disjoint sets of vertices and E is the set of edges:

- $|U| = |s_{AB}|$, each vertex is labeled as $U_{s_{AB}(x)}$ for all K values of x;
- $|V| = |s_{AC}|$, each vertex is labeled as $V_{s_{AC}(x)}$ for all K values of x;
- $e_{ij} = (U_i, V_j) \in E$ if and only if $i = s_{AB}(x)$ and $j = s_{AC}(x)$ for some x.

In essence, each vertex U_i (or V_i) represents the set of value x's that produce the same value $s_{AB}(x) = i$ (or $s_{AC}(x) = i$); and each edge $e_{ij} = (U_i, V_j)$ represents the set of value x's that produces the same pair of channel symbols $s_{AB}(x) = i$ and $s_{AC}(x) = j$. Figure 4.2 on page 93 shows the bipartite graph corresponding to the MEQ-AD(3,6) protocol we introduced in Section 4.6. Near the nodes U_i and V_i we show the set of value x's such that $s_{AB}(x) = i$ and $s_{AC}(x) = i$, respectively. The number near each edges is the corresponding value of that edge.

Let $|e_{ij}|$ be the size of the set of value x's corresponding to edge eij. We first argue that

Lemma C.1 *$|e_{ij}| = 1$ for all $e_{ij} \in E$. Hence $|E| = M$ and $|U| \times |V| \geq M$.*

Proof: Suppose to the contrary that there exists some $e_{ij} \in E$ with $|e_{ij}| \geq 2$. Then there must be two values x, x' such that $x \neq x'$, $s_{AB}(x) = s_{AB}(x')$ and $s_{AC}(x) = s_{AC}(x')$. Similar to the "fooling set" argument in [17], it is impossible for nodes B and C to tell the difference between the two input vectors (x, x, x) and (x', x, x); hence they will not be able to solve the MEQ-AD(3,M) problem, which leads to a contradiction. Then the first part of the lemma follows.

Since every edge represents one x, and there are M possible values of x, it follows that $|E| = K$. Also, in a simple bipartite graph, we always have $|U| \times |V| \geq |E|$. Thus $|U| \times |V| \geq K$. □

Since there is a one-to-one mapping from the set of input values $\{1, \cdots, K\}$ to the edges E, we will use the terms input value (x) and edge (e_{ij}) interchangeably. Now we prove the following theorem on the constraint of s_{BC}:

Lemma C.2 *$s_{BC}(x) \neq s_{BC}(x')$ if edges x and x' are adjacent or there is some other edge that is adjacent to both of them, in the bipartite graph $G(U, V, E)$.*

Proof: Consider any pairs x, x' such that $x \neq x'$ and $(U_{s_{AB}(x)}, V_{s_{AC}(x')}) \in E$. Let x^* (maybe equal to x or x') be the input value that edge $(U_{s_{AB}(x)}, V_{s_{AC}(x')})$ corresponds to. So we have $s_{AB}(x^*) = s_{AB}(x)$ and $s_{AC}(x^*) = s_{AC}(x')$. Now consider the input vector (x^*, x, x') at node A, B and C. Since $s_{AB}(x^*) = s_{AB}(x)$, node B cannot differentiate (x^*, x, x') from (x, x, x). So node B cannot detect the mismatch. Meanwhile, since $s_{AC}(x^*) = s_{AC}(x')$, node C can not differentiate (x^*, x, x') from (x', x', x') by just looking into the receives s_{AC}. So $s_{BC}(x)$ must be different from $s_{BC}(x')$; otherwise, the MEQ-AD problem is not solved. Then the lemma follows. □

Now we can conclude that the problem of designing s_{BC}, given functions $s_{AB}(\cdot)$ and $s_{AC}(\cdot)$, is equivalent to finding a distance-2 edge coloring for the corresponding bipartite graph $G(U, V, E)$. Furthermore, it should not be hard to see that any protocol P that solves MEQ-AD(3,M) is equivalent to a bipartite graph $G(U, V, E)$ together with a distance-2 coloring scheme W such that $|U| = |s_{AB}|$, $|V| = |s_{AC}|$, $|E| = M$, and $|W| = |s_{BC}|$, where $|W|$ denotes the number of colors in scheme W. Notice that

$$C(P) = \log_2 |s_{AB}| + \log_2 |s_{AC}| + \log_2 |s_{BC}| = \log_2(|s_{AB}| \times |s_{AC}| \times |s_{BC}|). \quad \text{(C.1)}$$

Then Theorem 4.2 follows.

According to Theorem 4.2, we can conclude that the problem of finding $C_{AD}(3, K)$ is equivalent to the problem of finding the minimum of $|U| \times |V| \times |W|$ for the bipartite graphs and distance-2 coloring schemes that satisfy the above constrains.

Using Theorem 4.2, to show that $C_{AD}(3, 4) = 4$, we only need to show that for every combination of $|U| \times |V| \times |W| < 2^4 = 16$ there exists no bipartite graph $G(U, V, E)$ and distance-2 coloring scheme W that satisfy the conditions as described in Theorem 4.2. In other words, if the conditions are all satisfied, then the bipartite graph $G(U, V, E)$ cannot be distance-2 colored with $|W|$ colors. It is not hard to see that there are only two combinations (up to permutation) that satisfy all conditions and have product less than 16: (2, 2, 2) and (2, 2, 3). Notice that in both cases, $|E| = |U| \times |V|$, where every pair of edges are within distance of 2 of each other, which means graph G can only be distance-2 colored with at least $|E|$ colors, which in this case is 4. Together with the upper bound from Section 4.4, this proves that $C_{AD}(3, 4) = C_{CD}(3, 4) = 4$.

Similarly, it can be shown that $C_{AD}(3, 6) = \log_2 27$. There are only two combinations that satisfy all conditions in Theorem 4.2 and have product less than 27: (2, 3, 3) and (2, 3, 4). Again, $|E| = |U| \times |V|$, so at least $|E| = 6$ colors are needed, which proves that $C_{AD}(3, 6) = \log_2 27$.

REFERENCES

[1] "Amazon S3 availability event," July 2008. [Online]. Available: http://status.aws.amazon.com/s3-20080720.html

[2] "Gmail disaster: Reports of mass email deletions," December 2006. [Online]. Available: http://techcrunch.com/2006/12/28/gmail-disaster-reports-of-mass-email-deletions

[3] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM (JACM)*, vol. 27, pp. 228–234, 1980.

[4] D. Dolev and R. Reischuk, "Bounds on information exchange for Byzantine agreement," *Journal of ACM (JACM)*, vol. 32, pp. 191–204, 1985.

[5] G. Liang and N. Vaidya, "Error-free multi-valued consensus with Byzantine failures," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 2011, pp. 11–20.

[6] G. Liang, B. Sommer, and N. Vaidya, "Experimental performance comparison of Byzantine fault-tolerant protocols for data centers," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[7] G. Liang and N. Vaidya, "Complexity of multi-valued Byzantine agreement," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., June 2010.

[8] G. Liang and N. Vaidya, "New efficient error-free multi-valued consensus with Byzantine failures," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., June 2011.

[9] G. Liang and N. Vaidya, "Brief announcement: Capacity of byzantine agreement with finite link capacity - complete characterization of four-node networks," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 2010.

[10] G. Liang and N. Vaidya, "Capacity of Byzantine agreement with finite link capacity," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 739–747.

[11] G. Liang and N. Vaidya, "Capacity of Byzantine consensus in capacity limited point-to-point networks," in *International Conference on COMmunication Systems and NETworkS (COMSNETS)*, 2012, pp. 1–10.

[12] G. Liang and N. Vaidya, "Capacity of Byzantine agreement: Complete characterization of the four node network," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., April 2010.

[13] N. Vaidya and G. Liang, "Capacity of Byzantine agreement (preliminary draft - work in progress)," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., January 2010.

[14] G. Liang and N. Vaidya, "Capacity of Byzantine agreement: Tight bound for the four node network," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., February 2010.

[15] G. Liang and N. Vaidya, "Capacity of Byzantine consensus with capacity limited point-to-point links," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., March 2011.

[16] G. Liang and N. Vaidya, "Byzantine broadcast in point-to-point networks using local linear coding," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., November 2011.

[17] A. C.-C. Yao, "Some complexity questions related to distributive computing (preliminary report)," in *ACM Symposium on Theory of Computing (STOC)*, 1979, pp. 209–213.

[18] G. Liang and N. Vaidya, "Multiparty equality function computation in networks with point-to-point links," in *SIROCCO*, 2010, pp. 258–269.

[19] G. Liang and N. Vaidya, "Multiparty equality function computation in networks with point-to-point links," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., October 2010.

[20] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2000, pp. 255–265.

[21] G. Liang, R. Agarwal, and N. Vaidya, "When watchdog meets coding," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2010, pp. 2267–2275.

[22] G. Liang and N. Vaidya, "When watchdog meets coding," Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Tech. Rep., May 2009.

[23] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1999, pp. 173–186.

[24] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative Byzantine fault tolerance," *ACM Transactions on Computer Systems*, vol. 27, pp. 7:1–7:39, 2010.

[25] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, pp. 398–461, 2002.

[26] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, "Fault-scalable Byzantine fault-tolerant services," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2005, pp. 59–74.

[27] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, "HQ replication: A hibrid quorum protocol for Byzantine fault-tolerance," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006, pp. 177–190.

[28] M. Fitzi and M. Hirt, "Optimally efficient multi-valued Byzantine agreement," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 2006, pp. 163–168.

[29] T. Xie and D. Feng, "How to find weak input differences for MD5 collision attacks," Cryptology ePrint archive, May 2009.

[30] S. Manuel, "Classification and generation of distrubance vectors for collision attacks against SHA-1," *Journal of Designs, Codes and Cryptography*, vol. 59, pp. 247–263, 2011.

[31] Z. Beerliova-Trubiniova and M. Hirt, "Perfectly-secure mpc with linear communication complexity," in *IACR Theory of Cryptography Conference (TCC)*, 2008, pp. 213–230.

[32] D. Dolev and H. R. Strong, "Authenticated algorithms for Byzantine agreement," *SIAM Journal on Computing*, vol. 12, pp. 656–666, 1983.

[33] B. Pfitzmann and M. Waidner, "Information-theoretic pseudosignatures and Byzantine agreement for $t \geq n/3$," IBM Research, Tech. Rep., 1996.

[34] P. Berman, J. A. Garay, and K. J. Perry, "Bit optimal distributed consensus," in *Computer science*. Plenum Press, 1992, pp. 313–321.

[35] B. A. Coan and J. L. Welch, "Modular construction of a Byzantine agreement protocol with optimal message bit complexity," *Journal of Information and Computation*, vol. 97, pp. 61–85, 1992.

[36] V. King and J. Saia, "Breaking the $O(n^2)$ bit barrier: Scalable Byzantine agreement with an adaptive adversary," in *ACM symposium on Principles of Distributed Computing (PODC)*, 2010, pp. 420–429.

[37] Z. Beerliova-Trubiniova and M. Hirt, "Efficient multi-party computation with dispute control," in *IACR Theory of Cryptography Conference (TCC)*, 2006.

[38] A. Patra and C. P. Rangan, "Communication optimal multi-valued asynchronous Byzantine agreement with optimal resilience," Cryptology ePrint Archive, 2009.

[39] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine fault tolerant systems tolerate Byzantine faults," in *USENIX Symposium on Netowrked Systems Design and Implementation (NSDI)*, 2009, pp. 153–168.

[40] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Transactions on Electronic Computers*, vol. EC-16, pp. 848–854, 1967.

[41] G. M. Masson, D. M. Blough, and G. F. Sullivan, "System diagnosis," in *Fault-Tolerant Computer System Design*. Prentice Hall, 1996.

[42] S. Mallela and G. Masson, "Diagnosable systems for intermittent faults," *IEEE Transactions on Computers*, vol. C-27, pp. 560–566, 1978.

[43] D. Blough and A. Pelc, "Complexity of fault diagnosis in comparison models," *IEEE Transactions on Computers*, vol. 41, pp. 318–324, 1992.

[44] R. W. Yeung and N. Cai, "Network error correction, part I: Basic concepts and upper bounds," *Communications in Information and Systems*, vol. 6, pp. 19–6, 2006.

[45] N. Cai and R. W. Yeung, "Network error correction, part II: Lower bounds," *Communications in Information and Systems*, vol. 6, pp. 37–54, 2006.

[46] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient network coding in the presence of Byzantine adversaries," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2007, pp. 616–624.

[47] D. Pradhan and N. Vaidya, "Roll-forward and rollback recovery: performance-reliability trade-off," *IEEE Transactions on Computers*, vol. 46, pp. 372–378, 1997.

[48] "Openssl project." [Online]. Available: http://www.openssl.org/

[49] M. J. Fischer, N. A. Lynch, and M. Merritt, "Easy impossibility proofs for distributed consensus problems," in *ACM symposium on Principles of Distributed Computing (PODC)*, 1985, pp. 59–70.

[50] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, 2003.

[51] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding (extended version)," 2004. [Online]. Available: http://www.its.caltech.edu/~tho/multicast.ps

[52] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Courier Dover Publications, 1998.

[53] A. K. Chandra, I. Merrick, L. Furst, and R. J. Lipton, "Multi-party protocols," in *ACM Symposium on Theory of Computing (STOC)*, 1983, pp. 94–99.

[54] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 2006.

[55] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transaction on Programming Languages and Systems*, vol. 4, pp. 382–401, 1982.

[56] E. Kushilevitz and E. Weinreb, "The communication complexity of set-disjointness with small sets and 0-1 intersection," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009, pp. 63–72.

[57] M. Pătraşcu and R. Williams, "On the possibility of faster SAT algorithms," in *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 1065–1075.

[58] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *ACM Symposium on Theory of Computing (STOC)*, 1996, pp. 20–29.

[59] Z. Bar-yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, "An information statistics approach to data stream and communication complexity," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002, pp. 209–218.

[60] A. Chakrabarti, S. Khot, and X. Sun, "Near-optimal lower bounds on the multi-party communication complexity of set disjointness," in *IEEE Conference on Computational Complexity (CCC)*, 2003, pp. 107–117.

[61] Y. Dinitz, S. Moran, and S. Rajsbaum, "Exact communication costs for consensus and leader in a tree," *Journal of Discrete Algorithms*, vol. 1, pp. 167–183, 2003.

[62] T. Ghosh, N. Pissinou, and K. Makki, "Towards designing a trusted routing solution in mobile ad hoc networks," *Journal of Mobile Networks and Applications*, vol. 10, pp. 985–995, 2005.

[63] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *Workshop on Mobile Computing Systems and Applications (WMCSA)*, 1999, pp. 90–100.

[64] S. Buchegger and J.-Y. Le Boudec, "Performance analysis of the CONFIDANT protocol," in *ACM International Symposium on Mobile ad hoc Networking & Computing (MobiHoc)*, 2002, pp. 226–236.

[65] C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas, "A quantitative trust establishment framework for reliable data packet delivery in MANETs," in *ACM Workshop on Security of Ad hoc and Sensor Networks (SASN)*, 2005, pp. 1–10.

[66] S. Ganeriwal and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," in *ACM Workshop on Security of Ad hoc and Sensor Networks (SASN)*, 2004, pp. 66–77.

[67] D. C. Kamal, D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *IEEE Conference on Information Sciences and Systems (CISS)*, 2006, pp. 857–863.

[68] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2008, pp. 1409–1417.

[69] F. Zhao, T. Kalker, M. Medard, and K. J. Han, "Signatures for content distribution with network coding," in *International Symposium on Information Theory (ISIT)*, 2007.

[70] Q. Li, D.-M. Chiu, and J. Lui, "On the practical and security issues of batch content distribution via network coding," in *IEEE International Conference on Network Protocols (ICNP)*, 2006, pp. 158–167.

[71] M. N. Krohn, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *IEEE Symposium on Security and Privacy*, 2004, pp. 226–240.

[72] C. Gkantsidis and P. Rodriguez Rodriguez, "Cooperative security for network coding file distribution," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2006, pp. 1–13.

[73] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," in *ACM Conference on Wireless Network Security (WiSec)*, 2009, pp. 111–122.

[74] M. Kim, M. Medard, and J. Barros, "Counteracting Byzantine adversaries with network coding: An overhead analysis," in *IEEE Military Communications Conference (MILCOM)*, 2008.

[75] M. Kim, R. Kotter, M. Medard, and J. Barros, "An algebraic watchdog for wireless network coding," in *International Symposium on Information Theory (ISIT)*, 2009.

[76] E. M. Palmer, "On the spanning tree packing number of a graph: a survey," *Journal of Discrete Mathematics*, vol. 230, pp. 13–21, 2001.

[77] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *International Symposium on Information Theory (ISIT)*, 2003.