The Communication Complexity of Fault-Tolerant Distributed Computation of Aggregate Functions

Yuda Zhao

A THESIS SUBMITTED
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2014

Acknowledgement

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this research. I would like to express my gratitude to all of them.

Foremost, I would like to express my sincere gratitude to my advisor Professor Haifeng Yu for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. He has been my inspiration as I hurdle all the obstacles during my entire period of Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Professor Seth Gilbert, Professor Rahul Jain and Professor Fabian Kuhn, for their encouragement, insightful comments, and suggestions to improve the quality of the thesis.

I thank my seniors Dr. Binbin Chen and Dr. Tao Shao, for their guidance and help in the last five years. I also thank my fellow labmates : Ziling Zhou, Xiao Liu, Feng Xiao, Padmanabha Seshadri, Xiangfa Guo, Chaodong Zheng, Mostafa Rezazad for all the fun we have had together.

Last but not the least, I would like to thank my family: My parents Wenhua Zhao and Ying Hu, for giving birth to me at the first place, taking care of me and supporting me spiritually throughout my life.

I am particularly grateful to my dearest Huang Lin for all the insightful thoughts and helping in the journey of life, proving her love and support during the whole course of this work.

Publication List

Results in this thesis are covered in following papers:

- Binbin Chen, Haifeng Yu, Yuda Zhao, and Phillip B. Gibbons. The Cost of Fault Tolerance in Multi-Party Communication Complexity. In *PODC*, July 2012. DOI=10.1145/2332432.2332442
- Binbin Chen, Haifeng Yu, Yuda Zhao, and Phillip B. Gibbons. The Cost of Fault Tolerance in Multi-Party Communication Complexity. In *JACM*, May 2014. DOI=10.1145/2597633
- Yuda Zhao, Haifeng Yu and Binbin Chen. Near-Optimal Communication-Time Tradeoff in Fault-Tolerant Computation of Aggregate Functions. In *PODC*, July 2014. DOI=10.1145/2611462.2611475

It should be noted that for the first two papers, the first three authors are alphabetically ordered.

Contents

Acknowledgement i

Publication List ii

Summary viii

1 Introduction 1

1.1 Background and Motivation 2

1.1.1 Communication Complexity 2

1.1.2 Fault-Tolerant Distributed Computation 3

1.1.3 Aggregate Functions 4

1.2 Our Goal 5

1.3 Related Work 5

1.3.1 SUM 5

1.3.2 Other Focuses in Fault-Tolerant Communication Complexity 7

1.3.3 Two-Party Communication Complexity 9

1.4 Our Contributions 10

1.4.1 The Exponential Gap Between the NFT and FT Communication Complexity of Sum 10

1.4.2 Near-Optimal Bounds on the zero-error FT Communication Complexity of General CAAFs 12

1.4.3 UnionSizeCP and the Cycle Promise 13

1.5 Organisation of the Thesis 14

2 Model and Definitions **15**

2.1 System Model 15

2.2 Commutative and Associative Aggregate Function 16

2.3 Time Complexity 17

2.4 NFT and FT Communication Complexity 18

2.5 Two-Party Communication Complexity 19

2.6 Some Useful Known Results 20

3 Upper Bounds on NFT Communication Complexity of Sum **23**

3.1 The Zero-Error Protocol 23

3.2 The (ϵ, δ)-Approximate Protocol 24

4 Lower Bounds on FT Communication Complexity of Sum for $b \leq N^{1-c}$ or $1/\epsilon^{0.5-c}$ **28**

4.1 Overview of Our Proof 29

4.1.1 UnionSize and UnionSizeCP 29

4.1.2 Overview of Our Reduction 31

4.2 Intuitions for Our Reduction from UnionSizeCP to Sum 32

4.3 A Formal Framework for Reasoning about Reductions to Sum . . . 34

4.4 Proof for Theorem 4.0.2 39

5 Communication Complexity of UnionSizeCP **43**

5.1 Alternative Form of the Cycle Promise 44

5.2 Zero Error Randomized Communication Complexity 45

5.2.1 Reduction from EqualityCP 46

5.2.2 Communication Complexity of EqualityCP 47

5.2.3 $O(\frac{n}{q}\log n + \log q)$ Upper Bound Protocol for UnionSizeCP . 49

5.3 (ϵ, δ)-Approximate Communication Complexity 50

5.3.1 Reduction from DisjointnessCP 50

5.3.2 Communication Complexity of DisjointnessCP 51

5.3.3 Proof for Theorem 5.3.1 57

6 The Fundamental Roles of Cycle Promise and UnionSizeCP **58**

6.1 Oblivious Reductions 59

6.2 The Completeness of UnionSizeCP 60

6.3 Proof for Theorem 6.2.1 61

6.4 Proof for Lemma 6.3.2 65

6.4.1 Node α and β Must Remain Unspoiled 67

6.4.2 Reasoning about Paths – Some Technical Lemmas 69

6.4.3 Proof for Lemma 6.4.1 81

7 Lower Bounds on FT Communication Complexity of Sum for All b **82**

7.1 Obtaining Some Intuitions under the Gossip Assumption 83

7.2 Topology and Adversary for Proving Theorem 1.4.2 84

7.3 The Probing Game and Its Connection to Sum 85

7.4 Lower Bound on the Number of Hits in the Probing Game 91

7.5 Proof for Theorem 1.4.2 94

8 Upper Bound on the FT communication complexity of general CAAFs **98**

8.1 Overview and Intuition 99

8.2 The Agg Protocol 101

8.2.1 Tree Construction/Aggregation and Some Key Concepts . . 102

8.2.2 Identify and Flood Potentially Blocked Partial Sums 104

8.2.3 Avoid Double Counting While Using Only Limited Information . . . 105

8.2.4 Pseudo-Code for The AGG Protocol . . . 107

8.2.5 Time Complexity and Communication Complexity of AGG . 107

8.2.6 Correctness Properties of AGG . . . 109

8.3 The VERI Protocol . . . 115

8.3.1 Design of The VERI Protocol . . . 116

8.3.2 Pseudo-Code for The VERI Protocol . . . 118

8.3.3 Time Complexity and Communication Complexity of VERI . 120

8.3.4 Correctness Properties of VERI . . . 120

8.4 Proof for Theorem 8.0.1 . . . 126

8.5 Dealing with Unknown f . . . 128

9 Conclusions and Future Work **130**

Bibliography **131**

Summary

Multi-party communication complexity involves distributed computation of a function over inputs held by multiple distributed players. A key focus of distributed computing research, since the very beginning, has been to tolerate failures. It is thus natural to ask "*If we want to compute a certain function while tolerating a certain number of failures, what will the communication complexity be?*"

This thesis centers on the above question. Specifically, we consider a system of N nodes which are connected by edges and then form some topology. Each node holds an input and the goal is for a special *root* node to learn some certain function over all inputs. All nodes in the system except the root node may experience *crash* failures, with the total number of edges incidental to failed nodes being upper bounded by f. This thesis makes the following contributions: 1) We prove that there exists an *exponential gap* between the non-fault-tolerant and fault-tolerant communication complexity of Sum; 2) We prove near-optimal lower and upper bounds on the fault-tolerant communication complexity of general *commutative and associative aggregates* (such as Sum); 3) We introduce a new two-party problem UnionSizeCP which comes with a novel *cycle promise*. Such a problem is the key enabler of our lower bounds on the fault-tolerant communication complexity of Sum. We further prove that this cycle promise and UnionSizeCP likely play a fundamental role in reasoning about fault-tolerant communication complexity of many functions beyond Sum.

List of Figures

1.1 The exponential gap between NFT and FT communication complexity of Sum 11

1.2 Summary of bounds on zero-error FT communication complexity of general CAAFs 13

4.1 The cycle promise for $q = 4$. 31

4.2 Lower bound topology for Sum 33

5.1 The alternative form of the cycle promise for $q = 4$ 45

6.1 Example assignment graph for a given node τ and for $b' = 4$ 61

6.2 Illustration of the 6 claims proved in Lemma 6.4.9 in an example topology 78

7.1 Example FT lower bound topology for $n = 4$ and unrestricted b. . . 83

8.1 Example aggregation tree and fragments. 104

8.2 Why speculative flooding is needed. 104

List of Tables

4.1 Key notations in Chapter 4. 30

6.1 Notations and definitions used in Section 6.4.2 and 6.4.3 70

8.1 Key notations in Chapter 8. 99

8.2 Guarantees of Agg and Veri under different scenarios. 116

Chapter 1

Introduction

This thesis considers a system consists of nodes which are connected by edges and then form some topology. Each node holds an input and the goal is for a special *root* node to learn some certain function over all inputs. All nodes in the system except the root node may experience *crash* failures. The fault-tolerant communication complexity of a function is defined as the least amount of communication required to compute the function, while tolerating failures. In comparison, the non-fault-tolerant communication complexity corresponds to the traditional setting where nodes are assumed to be failure-free. In this context, we have proved near-optimal lower and upper bounds on the fault-tolerant communication complexity of general *commutative and associative aggregates* (such as SUM). Coupled with some simple results, we actually have proved an *exponential gap* between the non-fault-tolerant and fault-tolerant communication complexity of SUM. Our results attests that fault-tolerant communication complexity needs to be studied separately from the simpler traditional non-fault-tolerant communication complexity, instead of being considered as an ”amended” version of non-fault-tolerant communication complexity. We've also introduced a new two-party problem UNIONSIZECP and further proved that the problem likely plays a fundamental role in reasoning about fault-tolerant communication complexity of many functions beyond SUM.

1.1 Background and Motivation

This thesis studies communication complexity of fault-tolerant distributed computation of aggregate functions. In the following sections, we will briefly review the concepts of communication complexity in Section 1.1.1, fault-tolerant distributed computation in Section 1.1.2, and aggregate functions in Section 1.1.3.

1.1.1 Communication Complexity

The notion of communication complexity was introduced by Yao [62] in 1979, and has been extensively studied after that. The original motivation arises from tasks in systems with multiple components: Any given single component in a system cannot *locally* perform a certain task if the task relies on data which are stored in other components. Determining the (least) amount of communication needed for various tasks is the central question of communication complexity theory. Communication complexity is related to many other areas such as VLSI circuits, data structures, streaming algorithms, and decision tree complexity. A comprehensive discussion of the techniques and applications of communication complexity can be found in Kushilevitz and Nisan's book [50].

Many models of communication complexity have been proposed. In following paragraphs, we will discuss models related to our work, from simplest ones to more complicated ones.

Two-party communication. Yao's two-party communication model is the first and simplest model in communication complexity. In this model, there are two parties named Alice and Bob. Each party holds an input string of n bits. The goal is to compute a certain function over their input strings. To achieve this, Alice and Bob have to communicate with each other. Among all the resources consumed in the process, communication complexity theory only focuses on the amount of communication between Alice and Bob — other factors such as the amount of local computation and the amount of memory required are ignored. This not only allows us to focus on the communication issue of the computation but also maps to applications properly — for example, in wireless sensor network, communication usually consumes far

more energy than local computation, and needs to be minimized for nodes operating on battery power [20].

Multi-party (blackboard) communication. The above two-party communication model has been naturally generalized to a model with more than two parties [17]. In the generalized model, multiple parties aim to compute a certain function over their inputs. Parties communicate by sending messages and each message is received by *all* parties. This communication model is equivalent to the case where all messages are written on a shared blackboard. For this reason, it is known as the *blackboard* model.

Multi-party (general topology) communication. The above blackboard model is not the only one for multi-party communication. In the *general topology* model, parties form some topology and may locally broadcast messages. Unlike in the blackboard model, here each message will only be received by the neighbors of the sender. This model can be viewed as an extension of above blackboard model — namely, the blackboard model is the special case where the topology is a clique.

We focus on the above general topology model instead of the blackboard model for reasons below. This thesis is motivated by distributed computation in large-scale wireless sensor networks and ad hoc networks. These networks consist of many low-cost nodes (sensors or wireless routers) distributed over a large physical area. Due to the limited wireless transmission range of these nodes, only nearby nodes can *directly* communicate with each other. This results in a multi-hop network topology that is often beyond the control of the protocol designer. For example, wireless sensor networks may be deployed simply by airplanes dropping sensors onto a target region [52], or deployed according to the specific physical environment that they monitor. The physical nature of these networks thus naturally requires one to consider general topologies.

1.1.2 Fault-Tolerant Distributed Computation

In many real distributed systems, components such as sensors may experience failures due to various reasons:

- Components may crash due to software issues such as application/OS/device driver crashes, deadlocks, and livelocks;
- Components may be compromised by malicious parties;
- Components may experience hardware failures;
- Communication links among components may fail permanently or temporarily due to various reasons such as being blocked by external objects.

In order to perform a given task in such a distributed system, a practical protocol should be robust to failures. Various failure models have been proposed for various distributed systems. We do not discuss them here since this thesis only focuses on *crash failures*. That means, we only consider the case where links are always reliable and components may crash. A component always exactly executes the given protocol until the protocol terminates or the component crashes. After that, it *never* executes any further operations. The notion of "failure" in this thesis, by default, refers to as crash failure.

1.1.3 Aggregate Functions

Formally, an *aggregate function* is a mapping from a set of values to a single value. Common aggregate functions include SUM (the sum of all inputs), AVG (the average value), MAX (the largest value), MIN (the smallest value), COUNT (the number of inputs), and etc. General *commutative and associative aggregate functions* (or CAAFs in short — see definition in Chapter 2) is a subset of aggregate functions including all above mentioned aggregate functions except AVG.

Distributed computation of aggregate functions is of fundamental importance in wireless sensor networks and wireless ad hoc networks. For example, consider a sensor network for temperature monitoring in a forest. In such a setting, the temperature reading of a single sensor often bears limited importance. Instead, we often need aggregate information such as the average temperature in a certain region [51]. This then corresponds to the computation of certain aggregate functions [35, 51] over the sensor readings.

1.2 Our Goal

Given a task of computing a certain aggregate function in a distributed system where components may fail, it is natural to ask how complicated the task is. In the context of communication complexity, it is asking "*If we want to compute a certain aggregate function in a fault-tolerant way, what will the communication complexity be?*". Such communication complexity of fault-tolerant distributed computing is referred to as fault-tolerant (FT) communication complexity in this thesis, while classical communication complexity which has been extensively studied is referred to as "non-fault-tolerant" (NFT) communication complexity. Taking account of failures leads to interesting questions:

- How big a difference can failures make in communication complexity?
- How does the number of failures affect communication complexity?

This thesis centers on the above questions.

1.3 Related Work

SUM is a key aggregate function in this thesis. Existing results on SUM will be provided in Section 1.3.1. As motivated in Section 1.1.1 and 1.1.2, this thesis focuses on tolerating crash failures in general topologies. Related to our focus, there have been prior works which focus on separate challenges such as privacy requirement and byzantine failures. These efforts are related to our work in the broader sense and will be discussed in Section 1.3.2.

1.3.1 SUM

The SUM function. Consider a synchronous network with N nodes and some undirected topology. Each node holds a binary value and the goal is for a special *root*

node to learn the sum of all inputs. (See Chapter 2 for a more formal description of the problem.)

Existing results on SUM. In failure-free settings, by leveraging in-network processing, a trivial tree-aggregation protocol can compute SUM with zero-error while requiring each node to send $O(\log N)$ bits. Since we consider general network topologies, we will naturally define communication complexity of a protocol as the number of bits sent by the bottleneck node instead of by all nodes combined (see Chapter 2 for formal discussion). Hence for zero-error results, the NFT communication complexity of SUM is upper bounded by $O(\log N)$. For (ϵ, δ)-approximate results, it is possible to further reduce to $O(\log \frac{1}{\epsilon} + \log \log N)$ bits per node for constant δ. In comparison, to tolerate arbitrary failures, there is a zero-error protocol for computing SUM which trivially having every node flood its id together with its value and thus requiring each node to send $O(N \log N)$ bits. To tolerate f edge failures (see Chapter 2 for formal definition), there is also a folklore SUM protocol that tolerates failures by repeatedly invoking the naive tree-aggregation protocol until it experiences a failure-free run. This protocol requires each node to send $O(f \log N)$ bits. For (ϵ, δ)-approximate results, researchers have proposed some protocols [5, 24, 53, 54, 65] where each node needs to send roughly $O(\frac{1}{\epsilon^2})$ bits for constant δ (after omitting logarithmic terms of $\frac{1}{\epsilon}$ and N). All these protocols conceptually map the value of each node to exponentially weighted positions in some bit vectors, and then estimate the sum from the bit vectors. Same as in one-pass distinct element counting algorithms in streaming databases [1, 28], doing so makes the whole process duplicate-insensitive. In turn, this allows each node to push its value along multiple directions to guard against failures. Note however, that duplicate-insensitive techniques do not need to be one-pass, and furthermore tolerating failures does not have to use duplicate-insensitive techniques. For example, one could repeatedly invoke the tree-aggregation protocol until one happens to have a failure-free run. There is also a large body of work [3, 12, 22, 21, 39, 42, 43] on computing SUM via gossip-based averaging (also called average consensus protocols). They all rely on the mass conservation property [43], and thus are vulnerable to node failures. There have been a few efforts [27, 40] on making these protocols fault-tolerant. However, they largely focus on correctness, without formal results on the protocol's communication complexity in the presence of failures. Despite all these efforts, no lower bounds on the FT communication complexity of SUM have ever been obtained, and thus it has been unknown

whether the existing protocols can be improved.

1.3.2 Other Focuses in Fault-Tolerant Communication Complexity

Secure multi-party computation. Our fault-tolerant communication complexity is related to the topic of secure multi-party computation [6, 7, 8, 10, 9, 18, 19, 29, 30, 34, 36, 37, 57, 63, 64]. Secure multi-party computation also aims to compute a function whose inputs are held by multiple distributed players. Different from our work, secure multi-party computation mainly focuses on the *privacy* requirement. Namely, when computing the function, a player should not learn any information about the inputs held by other players, except what can already be inferred from the output of the function. Research on secure multi-party computation usually investigates whether it is possible to compute a certain class of functions, and if yes, what is the communication complexity. The failure model considered by secure multi-party computation, given the security nature of the subject, is more diverse than our simple crash failure model. For example, researchers have considered players that i) are *curious* but follow the protocol [10, 29, 30, 34, 63, 64], ii) may crash [9, 29], or iii) may experience byzantine failures [6, 7, 8, 9, 10, 18, 19, 29, 30, 34, 36, 37, 57]. In terms of the topology among the players, to the best of our knowledge, research on secure multi-party computation almost always assumes that the players are fully connected and form a clique.

The central difference between our focus and secure multi-party computation is that the latter's key challenge is to preserve privacy. If privacy is not a concern, secure multi-party computation problems usually become trivial (i.e., with trivial and matching upper/lower bounds). In comparison, our focus does not concerned with privacy — the key challenge instead is to compute aggregate functions over general topologies (rather than just cliques). If we only consider cliques, most aggregate functions (such as Sum) becomes trivial (i.e., with trivial and matching upper/lower bounds).

Such a central difference between the two problems implies that they are incomparable — neither of them is easier than the other. Furthermore, upper bounds, lower

bounds, and proof techniques for one problem usually cannot carry over to the other. For example, the lower bounds in secure multi-party computation are usually derived from the privacy requirement, while we prove lower bounds on SUM by constructing proper lower bound topologies (i.e., worst-case topologies).

Communication complexity under unreliable channels. Other than in the topic of secure multi-party computation, tolerating node failures has not been considered in various developments on different models for communication complexity (e.g., [13, 17, 38, 58, 60]). Among these developments, the closest setting to tolerating node failures is perhaps unreliable channels [13, 31, 58, 60]. For example, the channels may flip the bits adversarially, flip each bit iid, or drop a certain number of messages. Under the iid unreliable channel model, there have also been some information-theoretic lower bounds on the rates of distributed computations [2, 33]. The specific techniques and insights for unreliable channels have limited applicability to tolerating node failures.

Bit complexity of other distributed computing tasks in failure-prone settings. Related to the computation of functions, distributed computing researchers have also studied the communication complexity (usually called *bit complexity* here) of other distributed computing tasks in failure-prone settings. For example, there has been a large body of work [23, 32, 45, 47, 44, 46, 56] on the bit complexity of distributed consensus and leader election. Compared to our work, all these efforts assume that the players are fully connected and form a clique. As explained earlier, for our focus, the key challenge is exactly to do the computation over general topologies instead of just cliques. On the other hand, these problems have their own unique challenges such as tolerating byzantine failures (instead of just tolerating crash failures as in our focus). Because of this, again, distributed consensus/leader election and our focus are incomparable — neither of them is easier than the other.

Some researchers feel that cliques may not be "realistic" topologies in some cases. Hence they explicitly construct low-degree network topologies, and then propose novel distributed consensus and leader election protocols specifically for those topologies [11, 48]. In some sense, the performance of these protocols are defined over the *best-case* topology that is low-degree. This corresponds to a setting where the topology is within the control of the protocol designer, and then a protocol is designed specifically for that topology. In comparison, as motivated in Section 1.1.1,

our focus considers general topologies where the performance (i.e., time complexity and communication complexity) of any given protocol is defined over the *worst-case* topology.

1.3.3 Two-Party Communication Complexity

Some of our results rely on the communication complexity of a novel two party problem UnionSizeCP introduced by us. Although UnionSizeCP has not been studied, it is related to some existing two-party problems.

The set disjointness problem. Disjointness is one of the most studied problems in two-party communication complexity. It is a binary function defined on two sets to test whether the two sets are disjoint. The function outputs 1 if and only if the two sets are disjoint. Otherwise, it outputs 0. Let n be the size of the universe where the two sets are generated. There is a trivial protocol where Bob sends all its input to Alice which enables Alice to determine the Disjointness function. This protocol leads to a trivial upper bound of $O(n)$. For deterministic protocols, there is a tight lower bound of $\Omega(n)$ [41]. The lower bound is proved by consider the rank of the *communication matrix* of the function. Each row of the communication matrix corresponds to a possible input of Alice's, and each column corresponds to a possible input of Bob's. An entry of the matrix is the Disjointness function over the corresponding input pair. For randomized protocols which can give the correct answer with a probability $2/3$ on every input, there is a tight lower bound of $\Omega(n)$ as well. A simple proof based on information theoretical approach appears in [4]. This approach is useful not only for Disjointness but also for our problems. See Section 5.3.2 for more details.

The gap Hamming distance problem. In this problem, Alice has a string from $\{0,1\}^n$ and so does Bob. Their goal is to determine whether the Hamming distance of the two strings is less than $n/2-\sqrt{n}$ or greater than $n/2+\sqrt{n}$. There is a trivial upper bound of $O(n)$. For deterministic protocols, by showing that the communication matrix does not contain a large *monochromatic rectangles* (defined in Section 5.2.2), a tight lower bound of $\Omega(n)$ can be proved. For randomized protocol which can give the correct answer with a probability $2/3$ on every input, researchers first consider

one-way protocols where only a single message is allowed. For these protocols, a linear lower bound on one-way communication complexity is proved in [61]. [14] further extends this lower bound to constant-round protocols. Finally, a tight lower bound of $\Omega(n)$ on the communication complexity of general protocols is proved in [16].

1.4 Our Contributions

This thesis centers on questions raised in Section 1.2. We have made following contributions: i) We have proved that there exists an *exponential gap* between the NFT and FT communication complexity of Sum, which will be discussed in Section 1.4.1; ii) We have proved near-optimal lower and upper bounds on the FT communication complexity of general CAAFs (defined in Chapter 2). Section 1.4.2 provides more detailed results; iii) We have introduced a new two-party problem UnionSizeCP which comes with a novel *cycle promise*. Such a problem is the key enabler of many results in this thesis. We have further proved that this cycle promise and UnionSizeCP likely play a fundamental role in reasoning about fault-tolerant communication complexity. Section 1.4.3 provides more discussions.

1.4.1 The Exponential Gap Between the NFT and FT Communication Complexity of Sum

As our first main contribution, we have proved an exponential gap between lower bounds on the FT communication complexity (or *FT lower bounds* in short) and upper bounds on the NFT communication complexity (or *NFT upper bounds* in short) of Sum. Our NFT upper bounds on Sum are obtained from well-known tree-aggregation protocol coupled with some standard tricks, which is not our main contribution. On the other hand, we have proved the first FT lower bounds on Sum for public-coin randomized protocols with zero-error and with (ϵ, δ)-error. Private-coin protocols and deterministic protocols are also fully but implicitly covered, and our exponential gap still applies. Our FT lower bounds are obtained for general f where f is an upper bound on the total number of edges incidental to failed nodes.

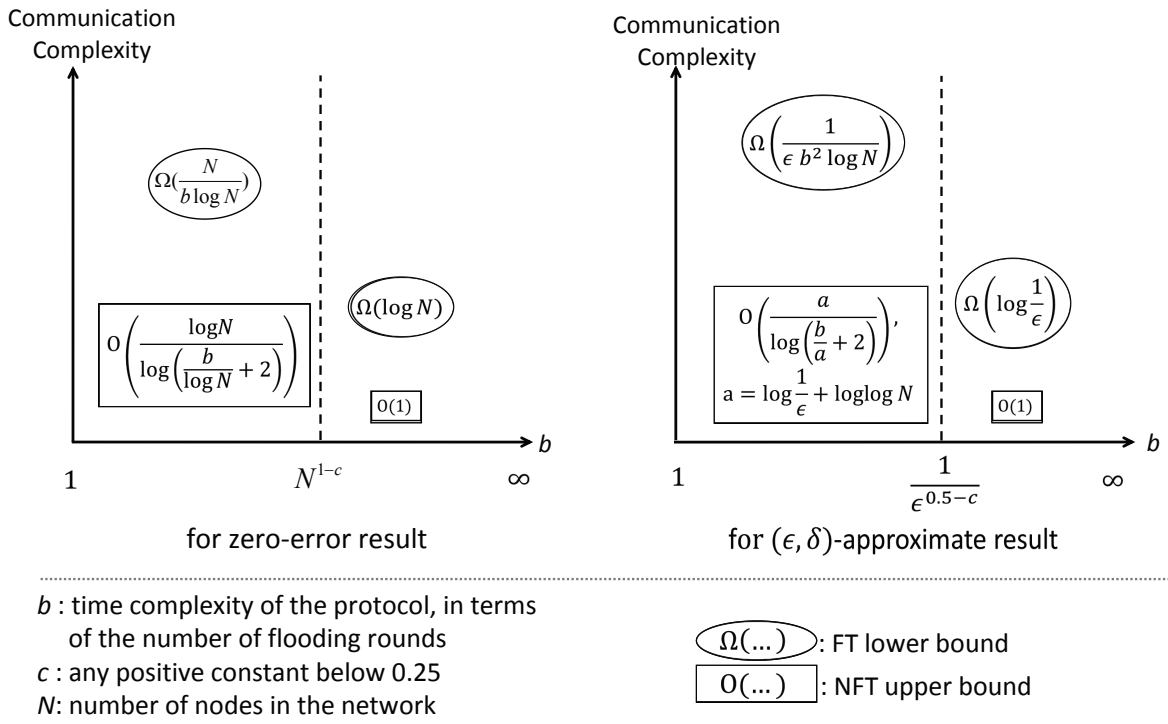

Figure 1.1: The exponential gap between NFT and FT communication complexity of Sum. All NFT upper bounds are obtained in Chapter 3. FT lower bounds (with $f = \Omega(N)$) are obtained in Chapter 4 (for $b \leq N^{1-c}$ or $\frac{1}{\epsilon^{0.5-c}}$), and Chapter 7 (for $b > N^{1-c}$ or $\frac{1}{\epsilon^{0.5-c}}$).

Nevertheless, in the following paragraph, we will only present our FT lower bounds in the case where $f = \Omega(N)$ since they are enough to show an exponential gap.

Since there is a tradeoff between communication complexity and time complexity, we always consider Sum protocols which can terminate within b *flooding rounds* (defined in Chapter 2), for b from 1 to ∞. Following theorem summarize our NFT upper bounds and will be proved in Chapter 3.

Theorem 1.4.1. *For any $b \geq 1$, we have:*

$$\mathcal{R}_0^{\text{syn}}(\text{Sum}_N, b) = O\left(a/\log\left(\frac{b}{a}+2\right)\right), \quad \text{where } a = \log N$$

$$\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn}}(\text{Sum}_N, b) = O\left(a/\log\left(\frac{b}{a}+2\right)\right), \quad \text{where } a = \log\frac{1}{\epsilon} + \log\log N$$

For fault-tolerant protocols, we have following Corollary 1.4.1 (proved in Chapter 4) and Theorem 1.4.2 (proved in Chapter 7).

Corollary 1.4.1. *For any $b \geq 1$, we have:*

$$\begin{aligned}
\mathcal{R}_0^{\text{syn,ft}}(\textsc{Sum}_N, N, b) &= \Omega\left(\frac{N}{b \log N}\right) \\
\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\textsc{Sum}_N, N, b) &= \Omega\left(\frac{1}{\epsilon b^2 \log N}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{N}}\right)
\end{aligned}$$

Theorem 1.4.2. *For any $b \geq 1$, we have:*

$$\begin{aligned}
\mathcal{R}_0^{\text{syn,ft}}(\textsc{Sum}_N, 2N, b) &= \Omega(\log N) \\
\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\textsc{Sum}_N, 2N, b) &= \Omega\left(\log \frac{1}{\epsilon}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{N}\right)
\end{aligned}$$

Figure 1.1 summarizes the exponential gap between the FT lower bounds and NFT upper bounds of Sum, which is established by the above 3 theorems. For $b \leq N^{1-c}$ or $\frac{1}{\epsilon^{0.5-c}}$ where c is any positive constant below 0.25, the NFT upper bounds are always at most logarithmic with respect to N or $\frac{1}{\epsilon}$, while the FT lower bounds are always polynomial.[1] For $b > N^{1-c}$ or $\frac{1}{\epsilon^{0.5-c}}$, the NFT upper bounds drop to $O(1)$, while the FT lower bounds are still at least logarithmic. Our results also imply that under small b values, the existing fault-tolerant Sum protocols (incurring $O(N \log N)$ or $O(\frac{1}{\epsilon^2})$ bits [5, 24, 53, 54, 65] per node) are actually optimal within polylog factors.

1.4.2 Near-Optimal Bounds on the zero-error FT Communication Complexity of General CAAFs

As our second main contribution, we have proved a novel upper bound of $O((\frac{f}{b}+1) \cdot \min(f \log N, \log^2 N))$ (Corollary 1.4.2, proved in Chapter 8) as well as a novel lower bound of $\Omega(\frac{f}{b \log b} + \frac{\log N}{\log b})$ (Corollary 1.4.3, proved in Chapter 4), for the zero-error FT communication complexity of general CAAF (such as Sum) protocols whose time complexity is within b flooding rounds (Figure 1.2). Note that our upper bound is no more than $O(\frac{f}{b} \log^2 N + \log^2 N)$, and hence is at most $\log^2 N \log b$ factor away from our lower bound.

[1] Here for (ϵ, δ)-approximate results, we only considered terms containing ϵ. Even if we take the extra terms with N into account, our exponential gaps continue to exist as long as $\frac{1}{\epsilon^c} = \Omega(\log N)$.

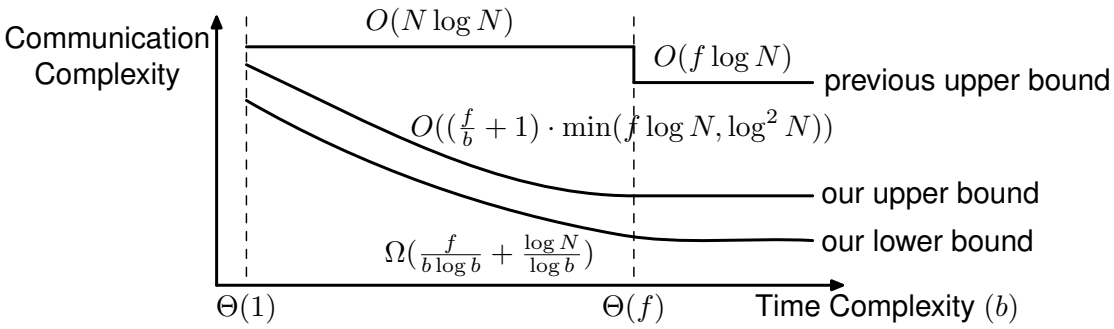

Figure 1.2: Summary of bounds on FT communication complexity of general CAAFs. Here b is the time complexity, and f is an upper bound on the total number of edges incident to failed nodes. Since the communication complexity depends on b, f, and N, the two-dimensional curves here are for illustration purposes only.

Corollary 1.4.2. *For any $b \geq 21c$ and $1 \leq f \leq N$,*

$$\mathcal{R}_0^{\text{syn,ft}}(\text{CAAF}_N, f, b) = O\left(\left(\frac{f}{b}+1\right)\cdot\min\left(f\log N, \log^2 N\right)\right)$$

Corollary 1.4.3. *For any $b \geq 1$ and $1 \leq f \leq N$, we have:*

$$\mathcal{R}_0^{\text{syn,ft}}(\text{CAAF}_N, f, b) \quad = \quad \Omega\left(\frac{f}{b\log b}+\frac{\log N}{\log b}\right)$$

Our upper bound protocol also, for the first time, allows a tunable tradeoff between communication and time complexity where the communication complexity can decrease polynomially with the time complexity. The protocol can also be easily extended (in Section 8.5) to settings with unknown f. Doing so will achieve a property similar to *early termination* — namely, the overhead of the protocol will automatically vary depending on the actual number of failures occurred during its execution.

1.4.3 UnionSizeCP and the Cycle Promise

Most of our FT lower bounds are obtained via an interesting reduction from a two-party communication complexity problem UnionSizeCP, where Alice and Bob intend to determine the size of the union of two sets, while the two sets satisfies a novel *cycle promise*. We further have found that UnionSizeCP and the cycle promise

likely play a fundamental role in reasoning about the FT communication complexity. Identifying this UnionSizeCP problem and the cycle promise is our third main contribution.

Specifically, we have proved a strong completeness result showing that UnionSizeCP is *complete* among the set of *all* two-party problems that can be reduced to Sum in the FT setting via *oblivious reductions* (defined in Chapter 6). Namely, we have proved that every problem in that set can be reduced to UnionSizeCP. Our proof also implicitly derives the cycle promise, thus showing that it likely plays a fundamental role in reasoning about the FT communication complexity.

1.5 Organisation of the Thesis

In the next, Chapter 2 describes our models and formal definitions. Chapter 3 presents the upper bounds on the NFT communication complexity of Sum. Chapter 4 proves the lower bounds on the FT communication complexity of Sum for $b \leq N^{1-c}$ and $\frac{1}{\epsilon^{0.5-c}}$. Proofs in Chapter 4 rely on our novel results on the communication complexity of UnionSizeCP which are proved in Chapter 5. Next Chapter 6 proves the completeness result for UnionSizeCP, showing that the polynomial dependency on b in Chapter 4's lower bounds might be inherent in Chapter 4's overall approach. Chapter 7 then uses a different approach to prove the lower bounds on the FT communication complexity of Sum for all b. Chapter 8 proves the upper bound on the FT communication complexity of Sum and general CAAFs. Finally, Chapter 9 draws the conclusions and proposes future work.

Chapter 2

Model and Definitions

This chapter describes the system model and formal definitions used throughout this thesis. We first introduce our system model in Section 2.1. *Commutative and associative aggregate functions* is a subset of aggregate function which includes all function studied in this thesis. Its formal definition appears in Section 2.2. Section 2.3 introduces the definition of *time complexity*, which will be used in defining *NFT and FT communication complexity* in Section 2.4. Finally, some results in classical two-party communication complexity are related to our study, which are described with related definitions in Section 2.5.

2.1 System Model

Network model. We consider a system consists of N nodes which are connected by some undirected network topology G. Each node has a unique id of $\log N$ bits (log in this thesis is always base 2). A node knows neither G nor its neighbors in G [1]. Node i has an integer *input* o_i, whose domain size is polynomial of N. The goal is for a special *root* node (whose id is known by all nodes) to learn a certain aggregate function over all these inputs. We consider a synchronous timing model where protocols proceed in *rounds*. Similar to the model in [49], here in

[1] Actually, our lower bounds hold even if the topology G (including the ids of the N nodes) is known to all nodes.

each round, each node first receives all the messages sent by its neighbors in the previous round. Next it does some local computation and then may choose to send (i.e., locally broadcast) a single message, which will be received by all its neighbors in the next round.

Failure model. All nodes in the system, except the root, may experience crash failures. A node that is disconnected from the root (i.e., has no path to the root) due to the failures of other nodes is also considered as failed. We consider only oblivious failure adversaries that adversarially decide beforehand (i.e., before the protocol flips any coins) which nodes fail at what time. For convenience and similar to [26], we also talk about *edge failures* — we say that an edge *fails*, iff at least one of its end points experiences a crash failure. We use f to denote an upper bound on the total number of edge failures, ranging from 1 to $\Theta(N)$.[2] Except in Section 8.5, we assume that f is known to the protocol.

2.2 Commutative and Associative Aggregate Function

A binary operator $\diamond$ is *commutative and associative* if for all operands o_1, o_2, and o_3, we have $o_1 \diamond o_2 = o_2 \diamond o_1$ and $(o_1 \diamond o_2) \diamond o_3 = o_1 \diamond (o_2 \diamond o_3)$. A function $\mathcal{F}$ is called a *commutative and associative aggregate function*, or *CAAF* in short, if i) there exists a commutative and associative binary operator $\diamond$ such that $\mathcal{F}(o_1, o_2, ..., o_N) = o_1 \diamond o_2 \diamond ... \diamond o_N$, and ii) the domain size of $o_{i_1} \diamond o_{i_2} \diamond ... \diamond o_{i_k}$ is at most polynomial with respect to N, for all $1 \leq k \leq N$ where i_1 through i_k are arbitrary distinct indices. The second requirement stems from the "aggregate" nature of the function – "aggregating" o_{i_1} through o_{i_k} should generate an output whose size is not too large. CAAF covers a wide range of common aggregate functions such as Sum and Count. Many other aggregate functions such as Average, Median, and Percentile can be reduced to CAAF. In particular, Median and Percentile can be solved by doing a binary search over the output domain, while invoking logarithmic number of Count's.

Zero-error and (ϵ, δ)-approximate results. In failure-free settings, both zero-error and (ϵ, δ)-approximate results are well-defined: Given a function and all inputs of

[2] Certain graphs may have more than $\Theta(N)$ edges. Our upper bound protocol also holds in these graphs. But we focus on f between 1 and $\Theta(N)$ which applies to all graphs.

parties, the function value over all inputs, denoted by s, is the zero-error result and any (random variable) $\hat{s}$ such that $\Pr[|\hat{s} - s| \geq \epsilon s] \leq \delta$ is a (ϵ, δ)-approximate result. In failure-prone settings, failures may cause some input values unavailable for all protocols. For example, if a party fails before the time of sending its first messages, its input value can never affect the result of any given protocol. To make our study meaningful, we allow the computation to ignore/omit the inputs held by those players that have failed (i.e., crashed) or been disconnected. For any given CAAF $\mathcal{F}$ (defined from any binary operator $\diamond$), following the same definitions from [5], a *zero-error result* of $\mathcal{F}$ is any result equals $\diamond_{o \in S} o$ for some S where $S_1 \subseteq S \subseteq S_2$ where S_1 is the set of inputs of nodes which have not failed or been disconnected from the root due to other nodes' failures, and S_2 is the set of inputs of all nodes. An (ϵ, δ)-*approximate result* of $\mathcal{F}$ is any $\hat{s}$ such that for some zero-error result s, $\Pr[|\hat{s} - s| \geq \epsilon s] \leq \delta$.

2.3 Time Complexity

With respect to a topology G, the *time complexity* of a (randomized) protocol describes the number of rounds needed for it to terminate, under the worst-case inputs of nodes in G, the worst-case failure adversary, and the worst-case coin flips. The shape of G has a large impact on time complexity. Hence we will always describe time complexity in terms of *flooding rounds*. Here each flooding round consists of d rounds, where d is G's diameter and is assumed to be known to the protocol. We use b to denote the time complexity in terms of flooding rounds (i.e., the total number of rounds would be bd).

At any given point of time between round 1 and round bd, let H be the same as G except that all the failed nodes and their incidental edges have been deleted. H's diameter may be larger or smaller than G. For a flooding round to remain meaningful in such a context, we assume that the failures do not substantially increase the network's diameter. Specifically, we assume that the diameter of H is no larger than $c \cdot d$, where c is some constant known to the protocol. [3]

[3] Our upper bound protocol critically relies on this assumption. As part of our future work, we are currently working on a new lower bound proof that aims to show the necessity of this requirement.

2.4 NFT and FT Communication Complexity

Communication complexity of a protocol. Classic multi-party communication complexity problems [50] usually consider the total number of bits sent by all players, since they usually use the blackboard model where the blackboard is the bottleneck. In our distributed computing setting with a topology G, as in other problems in such a setting, it is more natural to consider the number of bits sent by the bottleneck player — the energies of sensors are usually provided by batteries. The capacity of batteries relies on the bottleneck player. Given a randomized protocol, a topology G, a value assignment to the nodes in G, and a failure adversary (if failures are considered), define a_i to be the *expected* (with the expectation taken over coin flips in the protocol) number of bits that node i sends. The protocol's *average-case communication complexity under* G is defined as the largest a_i, across all value assignments of the nodes in G, all failure adversaries (if failures are considered), and all i's ($1 \leq i \leq N$). The protocol's *worst-case communication complexity under* G is similarly defined by considering worst-case coin flips instead of taking the expectation over the coin flips.

Public coins versus private coins. In a randomized protocol, players can "toss coins". Formally, there are some strings which are randomly generated and players can access these strings in the following way. If public coins are allowed, there is only one string and all nodes have access to the string. Otherwise, players can only use private coins which means each player has a string and can only access its own one. In this thesis, we allow public coins. By default, the notion of "coins" in this thesis refers to public coins. [4]

Communication complexity of Sum **(zero-error case).** We define $\mathcal{R}_0^{\text{syn}}(\text{Sum}, G, b)$ to be the smallest average-case communication complexity under G across all randomized Sum protocols that can generate, in a failure-free setting, a zero-error result on G within a time complexity of at most b flooding rounds. We similarly define $\mathcal{R}_0^{\text{syn,ft}}(\text{Sum}, G, f, b)$ across all Sum protocols which can additionally tolerate up to f edge failures, if these failures do not substantially increase the network's diameter (See section 2.3). Here note that length of a flooding round depends on G. For

[4] In fact, all results in this thesis hold if only private coins are allowed. The lower bounds trivially hold. For the upper bound, although our protocol uses public coins, it can be avoided as we shown in [66].

any given integer N, we define $\mathcal{R}_0^{\text{syn}}(\text{SUM}_N, b)$ to be the maximum $\mathcal{R}_0^{\text{syn}}(\text{SUM}, G, b)$ across all topology G's where G is connected and has exactly N nodes. Similarly we define $\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b)$.

Communication complexity of SUM ((ϵ, δ)-approximate case). For (ϵ, δ)-approximate case, we use the worst-case communication complexity for defining, which is standard practice [4, 50]. We define $\mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\text{SUM}, G, b)$, to be the smallest worst-case communication complexity under G across all randomized SUM protocols that can generate, in a failure-free setting, (ϵ, δ)-approximate result on G within a time complexity of at most b flooding rounds. We similarly define $\mathcal{R}_{\epsilon,\delta}^{\text{syn,ft}}(\text{SUM}, G, f, b)$ across all randomized SUM protocols which can additionally tolerate up to f edge failures, if these failures do not substantially increase the network's diameter (See section 2.3). For any given integer N, we define $\mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\text{SUM}_N, b)$ to be the maximum $\mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\text{SUM}, G, b)$ across all topology G's where G is connected and has exactly N nodes. Similarly define $\mathcal{R}_{\epsilon,\delta}^{\text{syn,ft}}(\text{SUM}_N, f, b)$.

2.5 Two-Party Communication Complexity

Some proofs in this thesis will also need to reason about the NFT communication complexity of some two-party problems. In such a problem Π, Alice and Bob each have an input X and Y respectively, and the goal is to compute the function $\Pi(X, Y)$. For all two-party problems in this thesis, we only require Alice to learn the final result. We will often use n to denote the size of Π, as compared to N which describes the number of nodes in G. The *communication complexity* of a randomized protocol for computing Π is defined to be either the average-case or worst-case (over random coin flips) number of bits sent by Alice and Bob combined. In the classic setting without synchronous rounds [50], similar as earlier, we define $\mathcal{R}_0(\Pi)$ ($\mathcal{R}_{\epsilon,\delta}(\Pi)$, respectively) to be the smallest average-case (worst-case, respectively) communication complexity across all randomized protocols that can generate a zero-error result ((ϵ, δ)-approximate result, respectively) for Π.

We will also need to consider a second setting with synchronous rounds, adapted from [38]. Here Alice and Bob proceed in synchronous rounds, where in each round Alice and Bob may simultaneously send a message to the other party. Alice, or Bob,

or both may also choose not to send a message in a round. The *time complexity* of a randomized protocol for computing Π is defined to be the number of rounds needed for the protocol to terminate, over the worst-case input and the worst-case coin flips. We define $\mathcal{R}_0^{\text{syn}}(\Pi, t)$ ($\mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\Pi, t)$, respectively) to be the smallest average-case (worst-case, respectively) communication complexity across all randomized protocols for Π that can generate a zero-error result ((ϵ, δ)-approximate result, respectively) within a time complexity of at most t rounds.

2.6 Some Useful Known Results

This section describes some known results that this thesis uses. These results and their proofs are not our contribution. We include the details and sometime the proofs here only for completeness, because some of them were folklore results, or were not formally stated, or were not stated to cover FT communication complexity, or were proved under slightly different models in a restricted form. In the next, the notations $\mathcal{R}_{0,\delta}$, $\mathcal{R}_{0,\delta}^{\text{syn}}$, and $\mathcal{R}_{0,\delta}^{\text{syn,ft}}$ simply mean $\mathcal{R}_{\epsilon,\delta}$, $\mathcal{R}_{\epsilon,\delta}^{\text{syn}}$, and $\mathcal{R}_{\epsilon,\delta}^{\text{syn,ft}}$ with $\epsilon = 0$, respectively.

Known relation between $\mathcal{R}_0$, $\mathcal{R}_0^{\text{syn,ft}}$ and $\mathcal{R}_{0,\delta}$, $\mathcal{R}_{0,\delta}^{\text{syn,ft}}$. Note that we do not necessarily have $\mathcal{R}_0 \geq \mathcal{R}_{0,\delta}$, since $\mathcal{R}_0$ is the average-case (over random coin flips in the protocol) communication complexity, while $\mathcal{R}_{0,\delta}$ is the worst-case (over random-coin flips in the protocol) communication complexity. Nevertheless, the following relation in Lemma 2.6.1 is well-known [50]. This relation trivially applies to fault-tolerant communication complexity as well.

Lemma 2.6.1. (Adapted from [50].) *For any communication complexity problem Π and $\delta > 0$, $\mathcal{R}_0(\Pi) \geq \delta\mathcal{R}_{0,\delta}(\Pi)$. Similarly for any $f \geq 1$, $b \geq 1$ and $\delta > 0$, $\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \geq \delta\mathcal{R}_{0,\delta}^{\text{syn,ft}}(\text{SUM}_N, f, b)$.*

Proof. Consider the optimal zero-error randomized protocol for Π, which generates a zero-error result while incurring an *expected* (over the random coin flips in the protocol) communication complexity of $\mathcal{R}_0(\Pi)$ bits. By Markov's inequality, the protocol's communication complexity exceeds $\mathcal{R}_0(\Pi)/\delta$ bits with probability at most δ. We can thus construct a new protocol which behaves the same as the original

one except that a node stops once it has sent $\mathcal{R}_0(\Pi)/\delta$ bits. Obviously, this protocol outputs correct results with probability at least $1-\delta$, and incurs a *worst-case* communication complexity of $\mathcal{R}_0(\Pi)/\delta$ bits, implying $\mathcal{R}_{0,\delta}(\Pi) \leq \mathcal{R}_0(\Pi)/\delta$. A similar proof can show $\mathcal{R}^{\text{syn,ft}}_{0,\delta}(\textsc{Sum}_N, f, b) \leq \mathcal{R}^{\text{syn,ft}}_{0}(\textsc{Sum}_N, f, b)/\delta$. □

Known relation between $\mathcal{R}_0$, $\mathcal{R}_{\epsilon,\delta}$ and $\mathcal{R}^{\text{syn}}_0$, $\mathcal{R}^{\text{syn}}_{\epsilon,\delta}$. The following lemma is a slightly extended version of the corresponding theorem from [38], which draws a connection between NFT communication complexity with synchronized rounds and NFT communication complexity without synchronized rounds. Since our synchronous round model is slightly different from [38], we provide a proof sketch below for the sake of completeness. This proof is not our contribution.

Lemma 2.6.2. (Adapted from [38].) *For any two-party communication complexity problem Π and any $t \geq 2$, we have $\mathcal{R}_0(\Pi) = \mathcal{R}^{\text{syn}}_0(\Pi, t) \cdot O(\log t)$ and $\mathcal{R}_{\epsilon,\delta}(\Pi) = \mathcal{R}^{\text{syn}}_{\epsilon,\delta}(\Pi, t) \cdot O(\log t)$.*

Proof. Consider any given protocol $\mathcal{P}$ (with $\mathcal{P}_A$ being Alice's part of the protocol and $\mathcal{P}_B$ being Bob's part), that can solve Π under the synchronous round setting with a bits (either on expectation or worst-case) of communication, while always terminating within t synchronous rounds. We construct a protocol $\mathcal{Q}$ (with $\mathcal{Q}_A$ and $\mathcal{Q}_B$ similarly defined) that can solve the problem with $O(a \log t)$ bits (either on expectation or worst-case, respectively) of communication complexity in the classic setting without synchronous rounds.

In $\mathcal{Q}$, Alice and Bob each maintains a local counter initialized to 1. These two counters correspond to the round number needed by $\mathcal{P}$. Let the current counter value on Alice be r_A. In $\mathcal{Q}_A$, Alice first *tries* executing $\mathcal{P}_A$ for rounds r_A, $r_A + 1$, $r_A + 2$, ..., while *assuming* that $\mathcal{P}_B$ does not send any message in any of those rounds. Alice then determines r'_A ($r'_A \geq r_A$), the first round during which $\mathcal{P}_A$ sends a message in this trial execution. Similarly Bob determines r'_B. Alice and Bob then exchange r'_A and r'_B, taking $2 \log t$ bits. Let $r' = \min(r'_A, r'_B)$. Alice next executes $\mathcal{P}_A$ (for real) for rounds r_A, $r_A + 1$, ..., r', and then sends a message to Bob if $r'_A = r'$. Similarly in $\mathcal{Q}_B$, Bob executes $\mathcal{P}_B$ for rounds r_B, $r_B + 1$, ..., r', and then sends a message to Alice if $r'_B = r'$. Note that for round r', $\mathcal{P}$ must incur at least one bit of communication. Thus for each bit $\mathcal{P}$ incurs, $\mathcal{Q}$ incurs at most $2 \log t + 1 = O(\log t)$ bits. After the message

exchange for round r', Alice and Bob set $r_A = r' + 1$ and $r_B = r' + 1$, and repeat the above process until $\mathcal{P}$ terminates. □

Chapter 3

Upper Bounds on NFT Communication Complexity of SUM

This chapter proves the following theorem, which describes the NFT upper bounds on SUM:

Theorem 1.4.1 (Restated)**.** *For any $b \geq 1$, we have:*

$$\mathcal{R}_0^{\text{syn}}(\text{SUM}_N, b) = O\left(a/\log\left(\frac{b}{a}+2\right)\right), \quad \textit{where } a = \log N$$

$$\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn}}(\text{SUM}_N, b) = O\left(a/\log\left(\frac{b}{a}+2\right)\right), \quad \textit{where } a = \log\frac{1}{\epsilon} + \log\log N$$

The above theorem is from well-known tree-aggregation protocols coupled with some standard tricks. These are not our main contribution — instead, they serve to show the exponential gap from our FT lower bounds. Its proof is obtained by combining following two sections.

3.1 The Zero-Error Protocol

In the protocol, nodes first construct a spanning tree and then aggregates all values from leaf nodes to the root. The spanning tree is simply constructed as follow:

Initially, the root broadcasts a special token. When a node A receives the token for the first time, A sets the sender B as its parent and informs B that A should be one of B's children. (If A has multiple candidate parents, to make everything deterministic, the candidate with the smallest id is chosen as A's parent.) In the next round, A broadcasts the token. Obviously, one round later A knows all its children. With this tree in place, a node becomes *ready* when it receives one *aggregation message* from each of its children. Each *aggregation message* encodes the *partial sum* of all the values in the corresponding subtree. Leaf nodes are *ready* when they knows they have no children. A ready node will combine all these aggregation messages, together with its own value, and then send a single aggregation message to its parent. Since each aggregation message uses $O(\log N)$ bits to encode the exact partial sum, the above protocol is a deterministic protocol for SUM with $O(\log N)$ communication complexity and $\Theta(1)$ flooding round time complexity.

One can further reduce the communication complexity if the time complexity is b flooding rounds with $b > 1$, since we can now spend b rounds in sending all the bits previously sent in one round. It is known [38] that an a-bit message sent in one round can be encoded using $a/\log\frac{b}{a}$ bits sent over b rounds, for $b \geq 2a$. To do so, one bit is sent every $\frac{b}{a} \cdot \log\frac{b}{a}$ rounds. Leveraging the round number during which the bit is sent, each such bit can encode $\log(\frac{b}{a} \cdot \log\frac{b}{a}) \geq \log\frac{b}{a}$ bits of information. Therefore we have $\mathcal{R}_0^{\text{syn}}(\text{SUM}_N, b) = O(a/\log(\frac{b}{a} + 2))$, where $a = \log N$.

3.2 The (ϵ, δ)-Approximate Protocol

The tree-aggregation protocol described in section 3.1 is already an (ϵ, δ)-approximate protocol. In the protocol, each node sends one aggregation message which uses $O(\log N)$ bits to encode the exact partial sum. It is possible to reduce the size of the aggregation message to $O(\log\frac{1}{\epsilon} + \log\log N)$ bits, using a simple private-coin protocol with similar tricks as in AMS synopsis [1].

Protocol intuition. First, we should note that directly encoding each partial sum with $O(\log\frac{1}{\epsilon} + \log\log N)$ bits using a floating-point-style representation will not actually work, due to underflow issues when sequentially adding many small numbers to a large number. Thus instead, we will apply a similar trick as AMS synopsis [1].

Intuitively in this protocol, each "1" value in the system is *flagged* with a certain probability. The system then uses the simple tree-aggregation protocol from Section 3 to determine the exact total number (sum) of such flagged "1" values. By properly adjusting the flagging probability, we can always ensure that this sum is no larger than $120/\epsilon^2$, and thus the size of the aggregation message will be no larger than $\log(120/\epsilon^2)$. Furthermore, it is possible to dynamically adjust such flagging probability in one pass of the aggregation protocol, without any global coordination. Finally, the root estimates the final result for SUM based on the sum of flagged "1" values and the associated flagging probability.

Algorithm 1 **promote**(msg)

```
msg.level + +;
Initialize tmp to 0;
for j = 1 to msg.sum do
    Increase tmp by 1 with probability 1/2;
end for
msg.sum = tmp;
```

Algorithm 2 **merge**(msg_1, msg_2) // assuming $msg_1.level \leq msg_2.level$

```
while msg1.level < msg2.level do
    promote(msg1);
end while
msg3.level = msg2.level;
msg3.sum = msg1.sum + msg2.sum;
while msg3.sum > 120/ε^2 do
    promote(msg3);
end while
return msg3;
```

Protocol description and pseudo-code. Specifically in this protocol, each aggregation message contains an integer $sum \in [0, 120/\epsilon^2]$ and an integer $level \in [0, \log N]$. Intuitively, these two integers mean that if each "1" value in the subtree is flagged with probability 2^{-level}, then the partial sum of the flagged values is sum. A node with a value of 1 generates an aggregation message with $sum = 1$ and $level = 0$, for its own value. Intermediate tree nodes will need to combine multiple aggregation messages into one. Without loss of generality, we only need to explain how to combine two aggregation messages msg_1 and msg_2 into one, where $msg_1.level \leq msg_2.level$. We *promote* (Algorithm 1) an aggregation message msg_1, by i) increasing $msg_1.level$ by one, and ii) tossing $msg_1.sum$ fair coins and then updating $msg_1.sum$ to be the

total number of heads we observe. To merge msg_1 and msg_2 into msg_3 (Algorithm 2), we first repeatedly promote msg_1, until $msg_1.level = msg_2.level$. We then set $msg_3.level = msg_2.level$, and $msg_3.sum = msg_1.sum + msg_2.sum$. If $msg_3.sum > 120/\epsilon^2$, we will again repeatedly promote msg_3 until the first time that $msg_3.sum \leq 120/\epsilon^2$. Finally, imagine that the root has a virtual parent and let msg be the aggregation message sent by the root to its virtual parent. The root will estimate the final sum to be $msg.sum \times 2^{msg.level}$.

Formal properties. It is obvious that the number of bits sent by each node in this protocol is $O(\log \frac{1}{\epsilon} + \log \log N)$. We next prove that the protocol does give us an $(\epsilon, 1/3)$-approximate result:

Theorem 3.2.1. *Consider any graph G with N nodes and any constant $\epsilon \in (0, 1]$. Let s denote the exact sum of the values of all the N nodes and $\hat{s}$ denote output of the above protocol. We have:*

$$\Pr[(1-\epsilon)s \leq \hat{s} \leq (1+\epsilon)s] \geq \frac{2}{3}$$

Proof. Consider the sequence of random variables $S_0, S_1, \ldots$, where $S_0 = s$ and S_{i+1} (for $i \geq 0$) is the number of heads observed when flipping a fair coin exactly S_i times. Furthermore, for generating S_{i+1}, the random process uses the same coin flip results as the protocol uses in promoting all messages with $level = i$ (i.e., at Line 4 of Algorithm 1). Let random variable L be the smallest integer such that $S_L \leq z$ where $z = \frac{120}{\epsilon^2}$. Let msg be the aggregation message sent by the root to its virtual parent. We claim that $msg.level = L$ and $msg.sum = S_L$. First, it is impossible for $msg.level < L$, since otherwise $msg.sum$ will be above z and thus the msg will be promoted by the root. Next if $msg.level > L$, it means that some node must have observed a message msg' whose level is L, and has further promoted msg'. But this is impossible since if $msg'.level = L$, then $msg'.sum \leq S_L \leq z$ by our definition of L. Now given that $msg.level = L$, we have $msg.sum = S_L$.

Let $l = \lfloor \log_2 \frac{3s}{4z} \rfloor$, and we have $2^l \in [\frac{3s}{8z}, \frac{3s}{4z}]$ and $2^{l+2} \in [\frac{3s}{2z}, \frac{3s}{z}]$. Since for all $i \geq 0$, S_i is a binomial random variable with parameter $(s, 2^{-i})$, we have

$$\mathrm{E}[S_l] = 2^{-l}s \geq \frac{4}{3}z \quad \text{and} \quad \mathrm{VAR}[S_l] \leq \frac{8}{3}z$$
$$\mathrm{E}[S_{l+2}] = 2^{-l-2}s \leq \frac{2}{3}z \quad \text{and} \quad \mathrm{VAR}[S_{l+2}] \leq \frac{2}{3}z$$

We claim that with probability at most $\frac{1}{4}$, $L \notin [l+1, l+2]$, since by Chebyshev's inequality:

$$\begin{aligned}\Pr[L \le l] &= \Pr[S_l \le z] \le \frac{24}{z} \le \frac{1}{5} \\ \Pr[L > l+2] &= \Pr[S_{l+2} > z] \le \frac{6}{z} \le \frac{1}{20}\end{aligned}$$

Denote $\mathcal{E}_i$ as the event $2^i S_i \notin [(1-\epsilon)s, (1+\epsilon)s]$, and we claim that for any $i \le l+2$, $\Pr[\mathcal{E}_i] \le \frac{1}{40}$. Since S_i is a binomial random variable with parameter $(s, 2^{-i})$, We have $\mathrm{E}[2^i S_i] = s$ and $\mathrm{VAR}[2^i S_i] \le 2^{2i} 2^{-i} s = 2^i s$. By Chebyshev's inequality, we have $\Pr[\mathcal{E}_i] = 1 - \Pr[2^i S_i \in [(1-\epsilon)s, (1+\epsilon)s]] \le \frac{2^i}{\epsilon^2 s} \le \frac{3}{z\epsilon^2} = \frac{1}{40}$. Next, denote $\mathcal{E}$ as the event that $\hat{s} \notin [(1-\epsilon)s, (1+\epsilon)s]$, or equivalently $2^L S_L \notin [(1-\epsilon)s, (1+\epsilon)s]$. We have:

$$\begin{aligned}\Pr[\mathcal{E}] &= \sum_i \Pr[L=i] \Pr[\mathcal{E}|L=i] \\ &= \sum_{i \in [l+1, l+2]} \Pr[L=i] \Pr[\mathcal{E}|L=i] + \sum_{i \notin [l+1, l+2]} \Pr[L=i] \Pr[\mathcal{E}|L=i] \\ &\le \Pr[L=l+1] \Pr[\mathcal{E}_{l+1}|L=l+1] + \Pr[L=l+2] \Pr[\mathcal{E}_{l+2}|L=l+2] \\ &\quad + \sum_{i \notin [l+1, l+2]} \Pr[L=i] \\ &\le \Pr[\mathcal{E}_{l+1} \text{ and } L=l+1] + \Pr[\mathcal{E}_{l+2} \text{ and } L=l+2] + \frac{1}{4} \\ &\le \Pr[\mathcal{E}_{l+1}] + \Pr[\mathcal{E}_{l+2}] + \frac{1}{4} \le \frac{1}{20} + \frac{1}{4} < \frac{1}{3}\end{aligned}$$

□

Apply the same trick as described at the end of section 3.1, we have $\mathcal{R}^{\mathrm{syn}}_{\epsilon, \frac{1}{3}}(\mathrm{SUM}_N, b) = O(a/\log(\frac{b}{a} + 2))$ where $a = \log\frac{1}{\epsilon} + \log\log N$.

Chapter 4

Lower Bounds on FT Communication Complexity of Sum for $b \leq N^{1-c}$ or $1/\epsilon^{0.5-c}$

By a reduction from a novel two party communication problem UnionSizeCP, this chapter proves following lower bounds on the fault-tolerant communication complexity of Sum:

Theorem 4.0.2. *For any $b \geq 1$ and $1 \leq f \leq N$, we have:*

$$\begin{aligned} \mathcal{R}_0^{\text{syn,ft}}(\text{Sum}_N, f, b) &= \Omega\left(\frac{f}{b\log b} + \frac{\log N}{\log b}\right) \\ \mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{Sum}_N, f, b) &= \Omega\left(\frac{1}{\epsilon b^2 \log N} + \frac{\log \frac{1}{\epsilon}}{\log b}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{f}}\right) \end{aligned}$$

It should be noted that i) The above theorem becomes trivial for $b \geq N$ and $b \geq 1/\epsilon^{0.5}$. For this reason, we will only use these lower bounds when $b \leq N^{1-c}$ or $b \leq 1/\epsilon^{0.5-c}$ for some constant c. For larger bs, we will use other techniques to obtain non-trivial lower bounds. These techniques and related results will later be discussed in Chapter 7; ii) The constant $1/5$ and the requirement of $\epsilon = \Omega\left(1/\sqrt{f}\right)$ comes from our results on UnionSizeCP. We can actually prove the theorem for any positive constant less than $1/4$. However, we cannot relax the requirement of $\epsilon = \Omega(1/\sqrt{f})$. More details will be discussed in Chapter 5.

The above theorem becomes the following corollary for $f = N$:

Corollary 1.4.1 (Restated). *For any $b \geq 1$, we have:*

$$\begin{aligned}\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, N, b) &= \Omega\left(\frac{N}{b \log N}\right)\\ \mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, N, b) &= \Omega\left(\frac{1}{\epsilon b^2 \log N}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{N}}\right)\end{aligned}$$

As lower bounds on SUM, Theorem 4.0.2 trivially applies to general CAAFs:

Corollary 1.4.3 (Restated). *For any $b \geq 1$ and $1 \leq f \leq N$, we have:*

$$\mathcal{R}_0^{\text{syn,ft}}(\text{CAAF}_N, f, b) = \Omega\left(\frac{f}{b \log b} + \frac{\log N}{\log b}\right)$$

In the following, Section 4.1 first gives an overview of our proof for this theorem, which is based on a reduction from UNIONSIZECP to SUM. In other words, given any oracle protocol for solving SUM, we will construct a protocol for solving UNIONSIZECP. Section 4.2 elaborates the concrete intuitions behind our reduction. The formal reasoning and proofs then follow: Section 4.3 develops a formal framework for our reasoning and and Section 4.4 proves Theorem 4.0.2 using the framework and our lower bounds on the communication complexity of UNIONSIZECP, which are proved in Chapter 5.

4.1 Overview of Our Proof

4.1.1 UNIONSIZE and UNIONSIZECP

One possible approach to achieve fault tolerance when computing SUM is for the nodes to simultaneously propagate their values along multiple directions. But doing so will lead to duplicates which must be addressed. Thus it is natural to consider a potential reduction from the two-party communication complexity problem UNIONSIZE, which was used for obtaining the optimal $\Omega(\frac{1}{\epsilon^2})$ lower bound on the space complexity of one-pass distinct element counting [61].

N	number of nodes in the topology G
d	diameter of the topology G
c	diameter of the topology never exceeds cd due to failures
f	upper bound on the number of edge failures
b	SUM protocol's time complexity, in terms of flooding rounds
n	size of UNIONSIZECP problem
q	parameter in the cycle promise of UNIONSIZECP
X	Alice's input in UNIONSIZECP
Y	Bob's input in UNIONSIZECP
α	a node in the topology G which can be simulated by Alice
β	a node in the topology G which can be simulated by Bob
$S_{A,X}(r)$	the set of all spoiled nodes at round r with respect to Alice's input X
$S_{B,Y}(r)$	the set of all spoiled nodes at round r with respect to Bob's input Y

Table 4.1: Key notations in Chapter 4.

Definition 4.1.1 (UNIONSIZE)**.** *In* UNIONSIZE$_n$*, Alice and Bob respectively have length-n binary strings X and Y. Let X_i and Y_i denote the ith bit of X and Y, respectively. Alice aims to determine* $|\{i \mid X_i \neq 0 \text{ or } Y_i \neq 0\}|$.

For reasons which will be clear later, we do not reduce from UNIONSIZE. Instead, we will introduce and reduce from a new two-party communication complexity problem called UNIONSIZECP. UNIONSIZECP is intuitively UNIONSIZE extended with a novel promise which we call the *cycle promise*. This promise is not constructed ad hoc — rather, we will later (in Chapter 6) see that it can be *derived*.

Definition 4.1.2 (UNIONSIZECP)**.** *In* UNIONSIZECP$_{n,q}$ *where $q \geq 2$, Alice and Bob respectively have length-n strings X and Y. The characters in the strings are integers in* $[0, q-1]$*. Let X_i and Y_i denote the ith character of X and Y, respectively. X and Y satisfy the following* cycle promise *where for all i: If $X_i = 0$, then Y_i must be* 0 *or* 1*; if $X_i = q-1$, then Y_i must be $q-2$ or $q-1$; if $0 < X_i < q-1$, then Y_i must be $X_i - 1$ or $X_i + 1$. Alice aims to determine* $|\{i \mid X_i \neq 0 \text{ or } Y_i \neq 0\}|$.

This promise is illustrated in Figure 4.1 as a bipartite *promise graph*, where values for X_i and Y_i are vertices and two values are connected by an edge if they satisfy the promise. Note that this promise graph is actually a cycle and symmetric for X and Y, which makes our later arguments easier. Same as in UNIONSIZE, the goal in UNIONSIZECP is for Alice to determine $|\{i \mid X_i \neq 0 \text{ or } Y_i \neq 0\}|$. When $q = 2$, UNIONSIZECP degrades to UNIONSIZE.

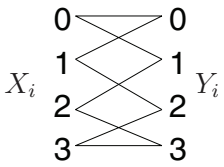

Figure 4.1: The cycle promise for $q = 4$.

4.1.2 Overview of Our Reduction

While the well-known reduction [61] from UNIONSIZE to the (centralized) one-pass distinct element counting problem is almost trivial, we seek a reduction from UNIONSIZECP to SUM. In particular, it is not immediately clear what a role failures can play. Our interesting reduction here will answer this question. Our reduction is based on a certain topology G. Given inputs X and Y to UNIONSIZECP, each node in G has some value so that their sum is exactly UNIONSIZECP(X, Y). The values of some of the nodes are uniquely determined by X, and thus are known by Alice from her local knowledge of X. If the value of a node τ cannot be uniquely determined by X, then τ is *spoiled* (rigorously defined in Section 4.3) for Alice, in the sense that Alice cannot simulate τ. As the simulation proceeds, a spoiled node τ may causally affect its neighbor node τ', rendering Alice unable to simulate τ' and thus making τ' spoiled as well. Since the SUM protocol may have internal state, if Alice cannot simulate a node for some round, then Alice cannot simulate the node for later rounds either. In this sense, a spoiled node can never get "unspoiled" later. For each round, Alice will simply simulate the (shrinking) group of all those nodes that have not been spoiled for Alice. Bob similarly simulates all unspoiled nodes for Bob. Alice's group and Bob's may intersect.

We want the root of G to remain unspoiled for Alice when the SUM protocol ends, so that it provides the SUM result to Alice for her to determine UNIONSIZECP(X, Y). To achieve this, in the reduction, Alice and Bob will need to strategically simulate the failures of certain nodes, to block the spreading of spoiled nodes. This showcases the fundamental role of failures in our reduction. At the same time, we need to avoid failing/disconnecting nodes with a value of 1 — failing/disconnecting them would enable the SUM protocol to ignore their values and potentially return a result that cannot be used to determine UNIONSIZECP(X, Y). (See correctness definition in Section 2.2.) In fact, if we were not concerned with this, then simply failing all nodes

except the root would keep the root unspoiled forever. Finally, it is also necessary to enlist help from Bob, who can simulate certain nodes that are spoiled for Alice. By forwarding to Alice messages sent by those nodes, Bob can further hinder the shrinking of Alice's group. The communication (between Alice and Bob) spent in doing so will be the communication complexity incurred for solving UNIONSIZECP. Simulating a shrinking group of nodes and properly using failures to hinder such shrinking is the main novelty in our reduction.

4.2 Intuitions for Our Reduction from UNIONSIZECP to SUM

This section intends to develop some concrete intuitions for our reduction from UNIONSIZECP to SUM. Given any SUM instance with parameter $b \geq 1$, $N \geq 14$, and $f \in [8, N]$, we map $\text{UNIONSIZECP}_{n,q}$ where $n = \lfloor \frac{f-2}{6} \rfloor$ and $q = 5b$ to the SUM problem on the topology in Figure 4.2. The topology has n parallel *chains* of nodes with 5 nodes on each chain. We connect one end of each chain to a node α, and the other end of each chain to a node β. We also connect α and β using an edge. We let α be the root of the topology. Finally, we connect $N - 5n - 4$ dummy nodes directly to β so that the total number of nodes (including the 2 dummy nodes described next) is exactly N. We further attach a chain of 2 dummy nodes to α so that the diameter of the initial topology is 5. Together with our design of the failure adversary later, these 2 dummy nodes will ensure that the diameter of the topology is always 5 in all rounds, meaning that the failures will not affect the diameter. This will make our lower bound construction as general as possible — more specifically, the construction will hold for all possible c ($c \geq 1$) values.

For all $1 \leq i \leq n$, consider the ith chain in the topology, and let the 5 nodes on the chain be v_1 through v_5, in order of increasing distance from α. The inputs X and Y to UNIONSIZECP will determine the values of v_3s. Specifically, the input to the middle node v_3 is 0 if $X_i = 0$ and $Y_i = 0$, otherwise the input to v_3 is 1. The inputs to all other nodes in the topology are always zero. X and Y also determine the failure time of other nodes. The node v_1 fails at the beginning of round $(X_i + 3)$ iff X_i is even, and v_4 fails at the beginning of round $(X_i + 3)$ iff X_i is odd. Similarly, the node v_5 fails at

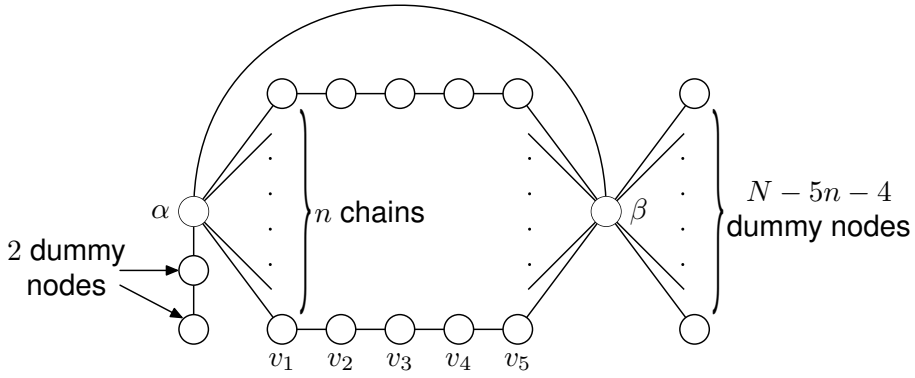

Figure 4.2: Lower bound topology for SUM. Since the topology can be viewed as a "distributed input" to the SUM problem, such a lower bound topology is analogous to a worst-case input commonly used for proving lower bounds.

the beginning of round $(Y_i + 3)$ iff Y_i is even, and v_2 fails at the beginning of round $(Y_i + 3)$ iff Y_i is odd. Finally, the 2 dummy nodes attached to α fail at the beginning of round 1. There are no other failures in the system. One can easily verify that the total number of edge failures (including edges incidental to disconnected nodes) is at most $6n + 2 \leq f$.

As a key property in the above construction, a v_3 whose value is 1 is never disconnected from the root. This is because if v_3's value is 1, then it must be unspoiled (by our construction) for either Alice or Bob, and thus can remain connected to α or β (and thus to the root). This in turn ensures that a zero-error result of SUM is always exactly UNIONSIZECP(X, Y).

In each round, Alice simulates the group of all the unspoiled nodes for Alice, including node α. Bob similarly simulates the unspoiled nodes for Bob, including node β. These two groups are made precise later in Section 4.4. Whenever α in the SUM protocol sends a message Alice always forwards that message to Bob. Bob does the same whenever β sends a message. Alice and Bob do not exchange any additional messages. Thus the number of bits sent by Alice and Bob for solving UNIONSIZECP is exactly the same as the number of the bits sent by α and β in the SUM protocol.

To see intuitively why this reduction works, consider Alice with input $X_i = 0$. Since Alice cannot determine v_3's value based on X_i, v_3 is spoiled for Alice. To prevent v_3 from causally affecting α and thus spoiling α, Alice simulates the failure of v_1 at (the beginning of) round 3 to block the influence of such v_3. Interestingly, Next since the failure of v_1 depends on X_i and is not uniquely determined by Y, the node

v_1 itself now becomes spoiled for Bob. With the cycle promise and since $X_i = 0$, Y_i must be 0 or 1. If $Y_i = 0$, then Bob does not need to be concerned, since Bob has already simulated the failure of v_5 at round 3 and thus blocked the potential influence of v_1.If $Y_i = 1$ however, Bob needs to simulate the failure of v_2 at round 4 to block the influence of v_1. Now v_2 again, becomes spoiled for Alice.Given the cycle promise and since $Y_i = 1$, we must have $X_i = 0$ or $X_i = 2$. If $X_i = 0$, then Alice has already simulated the failure of v_1 at round 3 and has already blocked the potential influence of v_2.If $X_i = 2$ however, Alice needs to simulate a new failure of v_1 at round 5.Extending such reasoning can show that by continuously injecting new failures, we can always manage to block the spreading of spoiled nodes.

Finally, note that the simulation still cannot continue forever. Under the cycle promise, it is possible for $X_i = Y_i = q - 1$. Thus we need the SUM protocol to stop by round $q + 1$, since otherwise at the beginning of round $q + 2$, Alice and Bob would simulate failures such that v_3 (with a value of 1) would be disconnected. This means that q needs to be chosen based on the SUM protocol's time complexity b: A larger q is needed when b is larger. Since the communication complexity of UNION-SIZECP depends on q (as shown later), as expected, our lower bounds here will be a function of b.

4.3 A Formal Framework for Reasoning about Reductions to SUM

Having provided the overview and intuitions, we are now ready for the formal reasoning. To facilitate our proof in Section 4.4 in the next, we develop a formal framework in this section.

Because FT communication complexity has not been formally studied before, many of the concepts in this framework need to be defined from scratch. A significantly simplified version of our framework, which does not involve failures, is implicitly used by Sarma et al. [59] for a reduction from the DISJOINTNESS two-party problem to distributed spanning tree verification. Applying the simplified version of our framework to their context would also streamline their proof.

Rounds and failures. The execution of the SUM oracle protocol starts at round 1. We sometimes for convenience also discuss round 0, during which the SUM protocol does nothing. Note that one can assume, without loss of generality, that all failures happen at the beginning of various rounds: If a node v fails sometime within round r, since v can (locally) broadcast at most one message in a round, the failure can be viewed as happening at the beginning of round $r+1$ if the failure occurs after v sends the message. Otherwise the failure can be viewed as happening at the beginning of round r. Thus from now on, we will assume that all failures happen at the beginning of various rounds. If a node fails at the beginning of round r, we say that the *failure time* of that node is round r.

Simulating a node. To properly *simulate* a node (i.e., simulate the execution of the SUM oracle protocol on that node) in a certain round, Alice (Bob) needs to feed all necessary parameters to the oracle protocol running on that node. The execution of a randomized oracle protocol on a node in a given round is uniquely determined by the (public) coin flips, the topology (since the topology is known), the id of the node, the (initial) value of the node, the failure time of the node (i.e., a failed node should not send out messages and the oracle protocol should not be invoked on such a node), and all the incoming messages to this node since round 1. Alice can easily generate the coin flips, and she already knows the topology and node id. For some nodes, Alice can uniquely determine their values and failure time based on Alice's input X. Finally, the incoming messages to a node v will have to be obtained via Alice's simulation of v's neighbors or directly from Bob if β is the sender of that message. Recall that in a round, a node first performs some local computation, and then does either a send or a receive. In particular, the potential message received by a node can only affect its behavior starting from the next round, since the node does not do any further local computation in the current round after the receive operation. Thus regardless of whether v does a send or receive in round r, to simulate v in round r, we (only) need all the incoming messages to v from round 1 to round $r-1$ (inclusive).

Epicenters and their occurrence time. The following concepts are always defined with respect to a given input X of Alice's. A node v in the topology G is a *value epicenter* if v's value is not uniquely determined by X. Namely given X, there exists Bob's inputs Y and Y' such that v's value is different under the simulated execution of the SUM oracle protocol (used in the reduction) for (X, Y) and (X, Y'). A node v

is a *failure epicenter* if v is not already a value epicenter and if v's failure time is not uniquely determined by X. Value epicenters and failure epicenter are all called *epicenters*.

The *occurrence time* of a value epicenter v is defined to be round 1. The *occurrence time* of a failure epicenter v is defined to be v's *earliest* failure time, across all valid Y's given the current X. To get some intuition behind the occurrence time, consider a failure epicenter v. Suppose that given X, the only possible inputs to Bob are Y and Y'. Imagine that v fails at the beginning of round 3 if Bob's input is Y, and fails at the beginning of round 8 if Bob's input is Y'. Since Alice does not know Bob's input, starting from round 3, Alice no longer knows whether v is still alive and thus can no longer simulate v. This also explains why a value epicenter v has an occurrence time of round 1 — Alice cannot simulate v even for round 1.

Spoil paths and spoiled nodes. All the following concepts are still with respect to a given input X of Alice's. If a node's failure time r is uniquely determined by X, we say that the node fails *stably* at the beginning of round r. We also call such a failure a *stable failure*. A *spoil path* from an epicenter u_0 (occurring at round r_0) to a node v is a sequence of nodes $u_0, u_1, u_2, ..., u_k, v$ where

- for $0 \leq i \leq k$, $u_i \neq \alpha$ and $u_i \neq \beta$,
- v is u_k's neighbor and u_i is u_{i-1}'s neighbor for $1 \leq i \leq k$,
- v has not failed stably before the beginning of round $r_0 + k + 2$, and u_i has not failed stably before the beginning of round $r_0 + i + 1$ for $0 \leq i \leq k$. Intuitively, this enables u_i to potentially send a message to u_{i+1} (and also u_{i+1} to receive this message) in round $r_0 + i + 1$. In turn, starting from round $r_0 + i + 2$, u_{i+1}'s behavior may potentially be affected by this message.

We define the *length* of a spoil path $u_0, u_1, u_2, ..., u_k, v$ to be $k + 1$. Intuitively, a spoil path is a potential path for u_0 to causally affect v, without going through α or β. Since Alice (Bob) will send to the other party all messages sent by α (β), paths going through α (β) are already taken care of. We intentionally define spoil paths in such a way that they can only be "blocked" by stable failures. This makes this definition consistent with the following intuition: If a node on a spoil path fails and

if that failure is not a stable failure, then that node must be an epicenter itself and will already cause the spreading of spoiled nodes. Thus intuitively, such a non-stable failure can never block the spreading of spoiled nodes. The *spoil distance* of a node v from an epicenter u_0 occurring at round r_0 is simply the length of the shortest spoil path from u_0 to v, or infinite if there is no such spoil path. For any epicenter u_0, we also define the spoil distance of u_0 from itself to be 0. For any given round r, a node v is *spoiled* in round r with respect to Alice's input X if v is within spoil distance of $r - r_0$ hops from some epicenter occurring at round r_0 where $r_0 \leq r$. By such definition, an epicenter with occurrence time of r becomes first spoiled in round r, which is consistent with the intuition. We use $S_{A,X}(r)$ to denote the set of all spoiled nodes at round r with respect to Alice's input X. We will prove in the next section that in each round r, Alice with input X can simulate all unspoiled nodes (i.e., all nodes in $\overline{S}_{A,X}(r)$).

We similarly define the notion of epicenters, spoil paths, spoiled nodes, and $S_{B,Y}(r)$ for Bob.

The simulatability lemma. Given the above formal framework, we can now prove the following simple but useful lemma which we will repeatedly invoke later.

Lemma 4.3.1. *Let X be Alice's input and Y be Bob's. Let R be any positive integer where $\alpha \in \overline{S}_{A,X}(R)$ and $\beta \in \overline{S}_{B,Y}(R)$. Assume that Alice (Bob) always forwards to the other party the message sent by α (β) in a round whenever Alice (Bob) is able to simulate the execution of the* SUM *oracle protocol on α (β) for that round. Then for all $0 \leq r \leq R$, Alice can properly simulate the execution of the* SUM *oracle protocol on all nodes in $\overline{S}_{A,X}(r)$ for round r and Bob can properly simulate all nodes in $\overline{S}_{B,Y}(r)$ for round r.*

Proof. We do an induction on r. First, $\alpha \in \overline{S}_{A,X}(R)$ and $\beta \in \overline{S}_{B,Y}(R)$ imply $\alpha \in \overline{S}_{A,X}(r)$ and $\beta \in \overline{S}_{B,Y}(r)$ for all $0 \leq r \leq R$. $\overline{S}_{A,X}(0)$ simply contains all nodes, since there are no epicenters occurring in round 0. Clearly, Alice can simulate all nodes for round 0 since the SUM protocol does nothing in round 0 and no failures happen in round 0. Similarly, for round 0 Bob can simulate all nodes in $\overline{S}_{B,Y}(0)$.

Assume that the claim holds for round r, and consider any node $v \in \overline{S}_{A,X}(r+1)$. We distinguish two cases:

- v is not an epicenter for Alice's input X. Then Alice can uniquely determine both the (initial) value and the failure time of v. If the failure time is round $r+1$ or earlier, then Alice trivially simulates v in round $r+1$ by doing nothing and we are done. Otherwise Alice knows that v is alive in round $r+1$.

- v is an epicenter for Alice's input X. We claim that it is impossible for the occurrence time of this epicenter to be round $r+1$ or earlier, since otherwise v would have been spoiled in round $r+1$ and thus would not be in $\overline{S}_{A,X}(r+1)$. Given that the occurrence time is round $r+2$ or later, it means that the occurrence time is not round 1. Thus v is not a value epicenter and Alice must know v's initial value. Furthermore, while Alice cannot determine v's exact failure time, Alice knows for sure that the failure time of v is round $r+2$ or later, and that v is alive in round $r+1$.

Now we only need to prove that Alice can simulate v in round $r+1$, given that Alice knows v's initial value and that v is alive in round $r+1$.

We trivially have $v \in \overline{S}_{A,X}(r)$ and by inductive hypothesis, Alice can simulate v from round 1 to r (inclusive). It thus suffices to prove that Alice can generate the potential message that v receives in round r from some neighbor u, so that Alice can simulate v in round $r+1$. We distinguish three cases for u. If u is β and since $\beta \in \overline{S}_{B,Y}(r)$, by inductive hypothesis, Bob can properly simulate β for round r. By condition of the lemma, Bob must have forwarded the message from β to Alice. Similarly if u is α and since $\alpha \in \overline{S}_{A,X}(r)$, Alice can properly simulate α for round r and generate the message herself. Finally, if $u \neq \alpha$ and $u \neq \beta$, we first show that u must be in $\overline{S}_{A,X}(r)$, via a contradiction. Since u sends a message in round r, u must have not failed in round r or earlier. In turn, u must have not failed stably in round r or earlier. Thus if $u \notin \overline{S}_{A,X}(r)$ (i.e., u is spoiled in round r), then v must be spoiled in round $r+1$, which contradicts with $v \in \overline{S}_{A,X}(r+1)$. Now given that $u \in \overline{S}_{A,X}(r)$, by inductive hypothesis Alice can simulate u for round r and generate u's message locally. Thus Alice has all the information needed to simulate v for round $r+1$.

Similar arguments apply to Bob. □

4.4 Proof for Theorem 4.0.2

We now move on to prove Theorem 4.0.2 using the framework established in Section 4.3 and a lower bound on UNIONSIZECP's communication complexity which will be proved in Chapter 5:

Theorem 4.4.1. *For any $t \geq 2$,*

$$\begin{aligned}
\mathcal{R}_0^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, t) &= \Omega\left(\frac{n}{q \log t}\right) - O\left(\frac{\log n}{\log t}\right) \\
\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, t) &= \Omega\left(\frac{1}{\epsilon q^2 \log t}\right) - O\left(\frac{\log \frac{1}{\epsilon}}{\log t}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{n}}\right)
\end{aligned}$$

Having Theorem 4.4.1, the proof of Theorem 4.0.2 is obtained by a series of lemmas. Lemma 4.4.1 below first proves that in the construction in Section 4.2, α (β) will always remain unspoiled for Alice (Bob) in round R, where R is large enough for the SUM oracle protocol to have terminated. Lemma 4.4.2 then proves the reduction from UNIONSIZECP to SUM. Combining this lemma with the earlier lower bound on UNIONSIZECP directly gives us Theorem 4.0.2.

Lemma 4.4.1. *Consider the topology and nodes (with their values and failure times) as constructed in Section 4.2. Under this construction and under $R = q + 1$, for all possible input X of Alice's, we have $\alpha \in \overline{S}_{A,X}(R)$. Similarly, for all possible input Y of Bob's, we have $\beta \in \overline{S}_{B,Y}(R)$.*

Proof. Without loss of generality, we prove $\alpha \in \overline{S}_{A,X}(R)$. We exhaustively consider all the epicenters with respect to Alice's input X. First, if $X_i = 0$ (implying Y_i must be 0 or 1), then v_3, v_2, and v_5 are the only epicenters on the ith chain. The spoil distance from all these epicenters to α is infinite, since v_1 fails stably at the beginning of round 3 and thus blocks the only possible spoil path.

Next if $X_i = q - 1$, then Y_i must be $q - 1$ or $q - 2$. If $q - 1$ is even, then v_2 (potentially occurring at round $q + 1$) and v_5 (potentially occurring at round $q + 2$) are the only epicenters on the ith chain. Again, v_1 fails stably at the beginning of round $q + 2$ and thus blocks the only possible spoil path from those two epicenters to α. If $q - 1$ is odd, then v_2 (potentially occurring at round $q + 2$) and v_5 (potentially occurring

at round $q+1$) are the only epicenters on the ith chain. Since v_1 fails stably at the beginning of round $q+2$, the only possible spoil path from v_5 to α is blocked. The epicenter of v_2 has an occurrence time of $q+2 > R$. Thus it can never cause α to be spoiled in round R.

Finally if X_i is even and $0 < X_i < q-1$, then Y_i must be odd and thus v_1 is the only epicenter on the ith chain, with an occurrence time of round X_i+2. (Recall that the occurrence time is the earliest possible failure time.) But since X_i is even, v_1 fails stably at the beginning of round X_i+3. This failure blocks the only possible spoil path from v_2 to α. The case where X_i is odd and $0 < X_i < q-1$ is similar. □

Lemma 4.4.2. *Consider any $b \geq 1$, $N \geq 14$, and $8 \leq f \leq N$. Let $n = \lfloor \frac{f-2}{6} \rfloor$ and $q = 5b$. We have*

$$\begin{aligned}
\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) &\geq \frac{1}{2}\mathcal{R}_0^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, 5b) \\
\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) &\geq \frac{1}{2}\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, 5b)
\end{aligned}$$

Proof. We construct G as in Section 4.2. It is easy to verify that in our construction, the failure adversary has introduced at most $6n+2 \leq f$ edge failures. And it is also easy to see the diameter of the topology is always 5 in all rounds.

Next we reduce the $\text{UNIONSIZECP}_{n,q}$ problem (in the synchronous rounds setting) to SUM. Consider any (black-box) oracle protocol for SUM. Given input X to Alice in UNIONSIZECP_n, Alice will simulate the execution of the oracle protocol on all nodes in $\overline{S}_{A,X}(r)$ at round r for $0 \leq r \leq R$. Similarly, given input Y to Bob, Bob simulates all nodes in $\overline{S}_{B,Y}(r)$. Furthermore, whenever α sends a message, Alice will forward that message to Bob. The same applies to Bob and β. Note that it is possible for Alice and Bob to send each other a message simultaneously in one round. By Lemma 4.3.1 and Lemma 4.4.1, such simulation is possible. When the oracle protocol terminates, which must be no later than round $5b \leq R$ since the time complexity of the oracle protocol is no larger than b flooding rounds, α and thus Alice will know the final result of the sum. By our construction, the zero-error result of the sum on G exactly equals $\text{UNIONSIZECP}(X, Y)$, and thus any (ϵ, δ)-approximate result of the sum is also an (ϵ, δ)-approximate result of $\text{UNIONSIZECP}(X, Y)$. The total amount of communication between Alice and Bob is exactly the total number of bits

sent by α and β combined in the above simulation. Thus either α or β must have sent at least half of the total number of bits sent by Alice and Bob. Thus we have:

$$\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \;\ge\; \mathcal{R}_0^{\text{syn,ft}}(\text{SUM}, G, f, b) \;\ge\; \frac{1}{2}\mathcal{R}_0^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, 5b)$$
$$\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, f, b) \;\ge\; \mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}, G, f, b) \;\ge\; \frac{1}{2}\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn}}(\text{UNIONSIZECP}_{n,q}, 5b)$$

□

The following proof for Theorem 4.0.2 follows naturally from the Lemma 4.4.2:

Theorem 4.0.2 (Restated)**.** *For any $b \ge 1$ and $1 \le f \le N$, we have:*

$$\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \;=\; \Omega\left(\frac{f}{b\log b} + \frac{\log N}{\log b}\right)$$
$$\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) \;=\; \Omega\left(\frac{1}{\epsilon b^2 \log N} + \frac{\log\frac{1}{\epsilon}}{\log b}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{f}}\right)$$

Proof. First, consider a trivial topology where the root has only a single neighbor A and all other nodes directly connect to A. Note that the domain size of SUM's output is $\Theta(N)$. In this topology, even if A already knows the SUM result, sending this result to the root within b flooding rounds still requires $\Omega(\frac{\log N}{\log b})$ bits (by Lemma 2.6.2). Similarly, if the root requires an $(\epsilon, \frac{1}{5})$-approximate result within b flooding rounds, node A has to send $\Omega(\frac{\log\frac{1}{\epsilon}}{\log b})$ bits. Hence there exists $c_1 > 0$ and $c_4 > 0$, such that

$$\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \;\ge\; \frac{c_1 \log N}{\log(5b)}$$
$$\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) \;\ge\; \frac{c_4 \log\frac{1}{\epsilon}}{\log(5b)}$$

For $f \in [1, 7]$ which implies $\epsilon = \Omega(1)$, we trivially have:

$$\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \;=\; \Omega\left(\frac{\log N}{\log b}\right) = \Omega\left(\frac{f}{b\log b} + \frac{\log N}{\log b}\right)$$
$$\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) \;=\; \Omega\left(\frac{\log\frac{1}{\epsilon}}{\log b}\right) = \Omega\left(\frac{1}{\epsilon b^2 \log N} + \frac{\log\frac{1}{\epsilon}}{\log b}\right)$$

Next we only need to consider $f \in [8, N]$. For sufficient large N, invoke Lemma 4.4.2 and Theorem 4.4.1 we have there exists positive constant c_2, c_3, c_5 and c_6 such that

$$\begin{aligned}
\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) &\ge \frac{1}{2}\mathcal{R}_0^{\text{syn}}(\text{UNIONSIZECP}_{\lfloor\frac{f-2}{6}\rfloor,5b}, 5b) \\
&\ge \frac{c_2\lfloor\frac{f-2}{6}\rfloor}{10b\log(5b)} - \frac{c_3\log\lfloor\frac{f-2}{6}\rfloor}{2\log(5b)} \\
&\ge \frac{c_2\lfloor\frac{f-2}{6}\rfloor}{10b\log(5b)} - \frac{c_3\log N}{\log(5b)} \\
\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) &\ge \frac{1}{2}\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn}}(\text{UNIONSIZECP}_{\lfloor\frac{f-2}{6}\rfloor,5b}, 5b) \\
&\ge \frac{c_5}{25\epsilon b^2\log(5b)} - \frac{c_6\log\frac{1}{\epsilon}}{\log(5b)}, \quad \text{for } \epsilon = \Omega\left(\frac{1}{\sqrt{f}}\right)
\end{aligned}$$

Combining with our earlier lower bound of $\frac{c_1\log N}{\log(5b)}$, we have

$$\begin{aligned}
\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) &= \frac{c_1}{2c_1+c_3}\cdot\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) + \frac{c_1+c_3}{2c_1+c_3}\cdot\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, f, b) \\
&\ge \frac{c_1}{2c_1+c_3}\cdot\left(\frac{c_2\lfloor\frac{f-2}{6}\rfloor}{10b\log(5b)} - \frac{c_3\log N}{\log(5b)}\right) + \frac{c_1+c_3}{2c_1+c_3}\cdot\frac{c_1\log N}{\log(5b)} \\
&= \frac{c_1c_2}{2c_1+c_3}\cdot\frac{\lfloor\frac{f-2}{6}\rfloor}{10b\log(5b)} + \frac{c_1^2}{2c_1+c_3}\cdot\frac{\log N}{\log(5b)} \\
&= \Omega\left(\frac{f}{b\log b} + \frac{\log N}{\log b}\right)
\end{aligned}$$

Similarly, we have

$$\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn,ft}}(\text{SUM}_N, f, b) = \Omega\left(\frac{1}{\epsilon b^2\log N} + \frac{\log\frac{1}{\epsilon}}{\log b}\right), \quad \text{for } \epsilon = \Omega\left(\frac{1}{\sqrt{f}}\right)$$

□

Chapter 5

Communication Complexity of UnionSizeCP

This chapter proves following Theorem 4.4.1, lower bounds on the communication complexity of UnionSizeCP:

Theorem 4.4.1 (Restated)**.** *For any $t \geq 2$,*

$$\begin{aligned}
\mathcal{R}_0^{\text{syn}}(\text{UnionSizeCP}_{n,q}, t) &= \Omega\left(\frac{n}{q \log t}\right) - O\left(\frac{\log n}{\log t}\right) \\
\mathcal{R}_{\epsilon,\frac{1}{5}}^{\text{syn}}(\text{UnionSizeCP}_{n,q}, t) &= \Omega\left(\frac{1}{\epsilon q^2 \log t}\right) - O\left(\frac{\log \frac{1}{\epsilon}}{\log t}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{\sqrt{n}}\right)
\end{aligned}$$

It should be noted that for the (ϵ, δ)-approximate case, i) the above theorem considers a constant δ of $1/5$. Techniques in this chapter can actually prove the same lower bound for any constant $\delta \in (0, 1/4)$. We cannot have larger constant since the above lower bound relies on Lemma 5.3.4 which is proved in [4]; ii) the requirement of $\epsilon = \Omega(1/\sqrt{n})$ comes from Theorem 5.3.2, which is obtained by a reduction from DisjointnessCP to UnionSizeCP. Unfortunately, DisjointnessCP is a binary function and we only have $(0, \delta)$-approximate results on its communication complexity. To draw a connection between $(0, \delta)$-approximate and (ϵ, δ)-approximate communication complexity, we have the above requirement on ϵ. We are not aware of any better techniques. It should be noted that since the range of UnionSizeCP is integer between 0 and n, we only need to consider $\epsilon \geq 1/n$. With this observation, we can

trivially have a slightly weaker lower bound for $\epsilon \in [1/n, 1/\sqrt{n}]$ as following

$$\begin{aligned}\mathcal{R}^{\text{syn}}_{\epsilon,\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}, t) &\geq \mathcal{R}^{\text{syn}}_{\frac{1}{\sqrt{n}},\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}, t) \\ &= \Omega\left(\frac{\sqrt{n}}{q^2 \log t}\right) - O\left(\frac{\log n}{\log t}\right) \\ &= \Omega\left(\frac{1}{\sqrt{\epsilon} q^2 \log t}\right) - O\left(\frac{\log \frac{1}{\epsilon}}{\log t}\right)\end{aligned}$$

Since UnionSizeCP has never been studied, there are no existing results on its communication complexity. Proving these results is thus also a contribution of our work, which may be of independent interest. This lower bound, has been used in the proof for Theorem 4.0.2 in Chapter 4. We prove it by directly combining Theorem 5.2.1 in Section 5.2, Theorem 5.3.1 in Section 5.3, and Lemma 2.6.2.

5.1 Alternative Form of the Cycle Promise

For the sake of convenience, we define the following alternative form of the cycle promise. As can be shown later, UnionSizeCP under two forms of the cycle promise are equivalent. In all other parts of this chapter (Section 5.2 and 5.3), we always consider problems under this alternative form of the cycle promise.

Definition 5.1.1 (Alternative Form of the Cycle Promise)**.** *Consider any two length-n strings X and Y where the characters in the strings are integers in* $[0, q-1]$*. X and Y satisfy the alternative form of the cycle promise iff for all i's where* $1 \leq i \leq n$*, either* $Y_i = X_i$ *or* $Y_i = (X_i + 1) \bmod q$*.*

Given X' and Y' satisfying the original cycle promise, Alice and Bob can always locally generate X and Y, such that X and Y satisfy the alternative form of the cycle promise and $\textsc{UnionSizeCP}(X, Y) = \textsc{UnionSizeCP}(X', Y')$. Specially to do so, Alice sets $X_i = X'_i/2$ for even X'_i and $X_i = q - (X'_i + 1)/2$ for odd X'_i. Bob sets $Y_i = (q - Y'_i/2) \bmod q$ for even Y'_i and $Y_i = (Y'_i + 1)/2$ for odd Y'_i. Clearly, we have $X_i = 0$ iff $X'_i = 0$, and $Y_i = 0$ iff $Y'_i = 0$, which implies that $\textsc{UnionSizeCP}(X, Y) =$

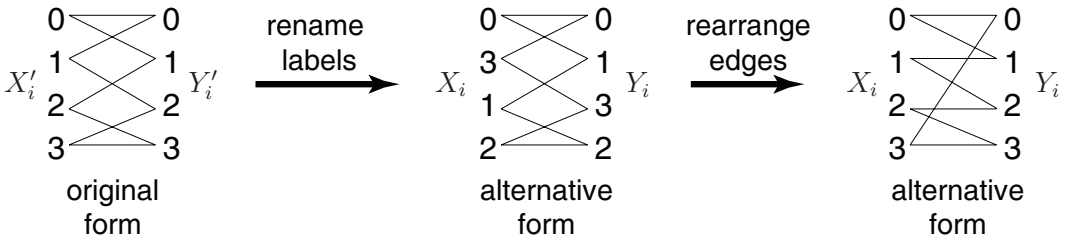

Figure 5.1: The alternative form of the cycle promise for $q = 4$, used *only* in Chapter 5 .

UNIONSIZECP(X', Y'). It is easy to verify that X and Y satisfy the alternative form of the cycle promise. Finally, since the above mapping from X' (Y') to X (Y) is a bijection, one can also construct a reverse mapping from X (Y) to X' (Y'). Given such mappings in both directions, we trivially know that the communication complexity of UNIONSIZECP with the original cycle promise is exactly the same as the communication complexity of UNIONSIZECP with the alternative form of the cycle promise.

5.2 Zero Error Randomized Communication Complexity

This section will prove the following lower bound on the zero-error randomized communication complexity of UNIONSIZECP:

Theorem 5.2.1. $\mathcal{R}_0(\text{UNIONSIZECP}_{n,q}) = \Omega(\frac{n}{q}) - O(\log n)$.

This lower bound on UNIONSIZECP is almost tight, given our upper bound $O(\frac{n}{q}\log n + \log q)$ upper bound later in Section 5.2.3. To obtain this lower bound, we introduce a new two-party problem called EQUALITYCP$_{n,q}$, which is the same as UNIONSIZECP$_{n,q}$ except that in EQUALITYCP$_{n,q}$, Alice and Bob aim to determine whether X equals Y. We are interested in EQUALITYCP$_{n,q}$ because its rectangular properties are easier to study. Following Section 5.2.1 constructs a reduction from EQUALITYCP to

UnionSizeCP. After that, Section 5.2.2 obtains a lower bound on the communication complexity of EqualityCP. Combining results in these two sections leads to the proof.

Proof (for Theorem 5.2.1). The theorem trivially holds for $n \leq q$. For $n > q$, combining Theorem 5.2.2 and 5.2.5 in following Section 5.2.1 and 5.2.2 directly yields the result. □

5.2.1 Reduction from EqualityCP

The following theorem establishes a reduction from EqualityCP to UnionSizeCP, based on the following observation: Knowing the result of UnionSizeCP, Alice and Bob can infer whether there exists j such that $X_j = q - 1$ and $Y_j = 0$. If there exists such j, then $X \neq Y$ and we are done. Otherwise for $1 \leq i \leq n$, we must have $Y_i = X_i$ or $Y_i = X_i + 1$ (note that there is no longer "mod q"). This implies that $X = Y$ iff $\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} Y_i$, making it easy to determine whether X equals Y.

Theorem 5.2.2. $\mathcal{R}_0(\text{EqualityCP}_{n,q}) \leq \mathcal{R}_0(\text{UnionSizeCP}_{n,q}) + O(\log q) + O(\log n)$.

Proof. We construct a reduction from EqualityCP to UnionSizeCP. To solve EqualityCP, Alice and Bob first invoke the oracle UnionSizeCP protocol on their inputs X and Y. Bob next sends Alice $\sum_{i=1}^{n} Y_i$, using $\log n + \log q$ bits, and the occurrence count (denoted as z) of the character 0 in Y, using $\log n$ bits. Alice finally outputs that X equals Y iff $\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} Y_i$ and UnionSizeCP(X, Y) equals $n - z$.

To show the correctness of the above protocol, note that if $X = Y$, then the two conditions trivially hold. We next prove the reverse direction. Since UnionSizeCP$(X, Y) = n - z$, for all i where $Y_i = 0$, we have $X_i = 0$. In turn, there does not exist i such that $X_i = q - 1$ and $Y_i = 0$. With this additional property, together with the cycle promise, we know that for $1 \leq i \leq n$, either $Y_i = X_i$ or $Y_i = X_i + 1$ (note that there is no longer "mod q"). Hence X must equal to Y since otherwise $\sum_{i=1}^{n} Y_i$ would be larger than $\sum_{i=1}^{n} X_i$. □

5.2.2 Communication Complexity of EqualityCP

This section proves Theorem 5.2.5, a lower bound on the communication complexity of EqualityCP. We first review some related concepts and results in communication complexity [50].

Definition 5.2.1 (Rectangle)**.** *A combinatorial rectangle (in short, a rectangle) in $X \times Y$ is a subset $R \subset X \times Y$ such that $R = A \times B$ for some $A \subset X$ and $B \subset Y$.*

Definition 5.2.2 (Monochromatic)**.** *With respect to a (partial) function f which is defined on a domain D, a rectangle R is monochromatic if f is fixed on $R \bigcap D$.*

Definition 5.2.3 (Cover number)**.** *For any (partial) function f, $C^1(f)$ is defined as the number of monochromatic rectangles needed to cover (possibly with intersections) the 1-inputs of f.*

Definition 5.2.4 (Nondeterministic communication complexity)**.** *For any (partial) function f, its nondeterministic communication complexity $N^1(f)$ is the size of the smallest string that can be used to convince both Alice and Bob that $f(x, y) = 1$. Formally $N^1(f)$ is the smallest integer k such that there exists function A and B, for all (x, y) in the domain of f,*

$$f(x, y) = 1 \Rightarrow \exists z \in \{0, 1\}^k : A(x, z) = 1 \wedge B(y, z) = 1$$
$$f(x, y) = 0 \Rightarrow \forall z \in \{0, 1\}^k : A(x, z) = 0 \vee B(y, z) = 0$$

For any partial function, its cover number is related to the communication complexity, as stated in following theorem.

Theorem 5.2.3. (Adapted from [50].) *For any (partial) function f, $\mathcal{R}_0^{pri}(f) \geq \log C^1(f)$.*

Proof sketch. $\mathcal{R}_0^{\text{pri}}(f) \geq N^1(f)$ holds since a nondeterministic protocol can "guess" random choices for which the computation ends within at most the average cost. $N^1(f) \geq \log C^1(f)$ holds since we can divide all (x, y) pairs where $f(x, y) = 1$ into groups based on their corresponding z. Each group is covered by a monochromatic rectangle and the number of groups is bounded by $2^{N^1(f)}$. Hence we have $2^{N^1(f)} \geq C^1(f)$. [1] □

[1] In fact, a stronger result that $2^{N^1(f)} = C^1(f)$ can be proved.

Next we apply an existing strong result on the Sperner capacity of directed graphs [15] to obtain a lower bound on the communication complexity of EqualityCP. That result was originally stated in the context of a directed coding graph, and the following instantiates it in our specific context:

Theorem 5.2.4. (Adapted from Theorem 3.2 in [15].) *Let S be a subset of $\{0,1,2,...,q-1\}^n$ with the following property: For all $V, W \in S$ where $V \neq W$, there i) exists i such that $V_i \neq W_i$ and $V_i \neq (W_i + 1)$ mod q, and ii) exists j such that $W_j \neq V_j$ and $W_j \neq (V_j + 1)$ mod q. Then $|S| \leq (rank(\mathbb{M}))^n$ for any $q \times q$ matrix $\mathbb{M}$, where $\mathbb{M}_{i,i} = 1$ for all i, $\mathbb{M}_{i,j} = 0$ for all $(j - i)$ mod $q \in \{2,3,...,q-1\}$, and all other entries in $\mathbb{M}$ (i.e., $\mathbb{M}_{1,2}$, $\mathbb{M}_{2,3}$, ..., $\mathbb{M}_{q-1,q}$, and $\mathbb{M}_{q,1}$) can be arbitrary real numbers.*

Theorem 5.2.5. $\mathcal{R}_0(\text{EqualityCP}_{n,q}) = \Omega(\frac{n}{q} - \log n - \log\log q)$.

Proof. Our definition of $\mathcal{R}_0$ allows public coins and only requires Alice to know the result. We define $\mathcal{R}_0^{\text{pri}}$ to be the same as $\mathcal{R}_0$ except that only private coins are allowed and both Alice and Bob are required to know the result. Using arguments based on rectangles [50], Lemma 5.2.1 next proves that $\mathcal{R}_0^{\text{pri}}(\text{EqualityCP}_{n,q}) \geq \frac{n}{q-1}$. The theorem follows since i) only one bit is needed for Alice to inform Bob the result, and ii) a public coin protocol using k bits here can always be simulated via private coins while using $O(k+\log\log(q^n \cdot q^n)) = O(k+\log n+\log\log q)$ bits [55]. □

Lemma 5.2.1. $\mathcal{R}_0^{\text{pri}}(\text{EqualityCP}_{n,q}) \geq \frac{n}{q-1}$.

Proof. According to Theorem 5.2.3, we only need to reason about the smallest number of monochromatic rectangles needed to cover all the 1-entries in the matrix corresponding to EqualityCP. The matrix $\mathbb{Z}$ corresponding to $\text{EqualityCP}_{n,q}$ is a $q^n \times q^n$ matrix. All 1-entries in $\mathbb{Z}$ are on the main diagonal. The remainder of $\mathbb{Z}$ consists of 0-entries and undefined entries that correspond to input pairs not satisfying the cycle promise. In any given covering of all the 1-entries using monochromatic rectangles, consider any two 1-entries $\mathbb{Z}_{V,V}$ (i.e., the entry for $X = V$ and $Y = V$) and $\mathbb{Z}_{W,W}$ in any rectangle used in the covering. For the rectangle to be monochromatic, $\mathbb{Z}_{W,V}$ and $\mathbb{Z}_{V,W}$ must not be 0-entries and hence must be undefined entries. This means that there i) exists i such that $V_i \neq W_i$ and $V_i \neq (W_i + 1)$ mod q, and ii) exists j such that $W_j \neq V_j$ and $W_j \neq (V_j + 1)$ mod q.

Applying Theorem 5.2.4 tells us that the number of 1-entries in such a monochromatic rectangle is upper bounded by $(rank(\mathbb{M}))^n$ for any $q \times q$ matrix $\mathbb{M}$ satisfying the properties specified in Theorem 5.2.4. We want to find such an $\mathbb{M}$ with a small rank, by properly choosing the values of $\mathbb{M}_{1,2}$, $\mathbb{M}_{2,3}$, ..., $\mathbb{M}_{q-1,q}$, and $\mathbb{M}_{q,1}$. We set all of them to be -1. We claim that the rank of such an $\mathbb{M}$ is exactly $q-1$. To see why, note that adding up all the q rows gives us an all-zero row, and hence $rank(\mathbb{M}) \leq q-1$. On the other hand, the first $q-1$ rows are linearly independent since $\mathbb{M}_{1,q} = \mathbb{M}_{2,q} = ... = \mathbb{M}_{q-2,q} = 0$ while $\mathbb{M}_{q-1,q} \neq 0$. Hence $rank(\mathbb{M}) = q-1$, implying that the number of 1-entries in a monochromatic rectangle of $\mathbb{Z}$ is upper bounded by $(q-1)^n$. Finally, because the total number of 1-entries in $\mathbb{Z}$ is q^n, we have $\mathcal{R}_0^{\text{pri}}(\text{EQUALITYCP}_{n,q}) \geq \log(q^n/(q-1)^n) = n\log(1+\frac{1}{q-1}) \geq \frac{n}{q-1}$. □

5.2.3 $O(\frac{n}{q}\log n + \log q)$ Upper Bound Protocol for UNIONSIZECP

In this (deterministic) protocol, given input X to Alice, let j ($0 \leq j \leq q-1$) be the integer with the smallest occurrence count in X. (If there are multiple such j's, simply pick an arbitrary one.) Alice first sends Bob the value of j and the set $Z = \{i \mid X_i = j\}$. This takes at most $O(\log q + \frac{n}{q}\log n)$ bits. Now we only need to worry about indices not in the set. For those indices, the promise graph (Figure 5.1) degrades to a chain, since two edges are removed from the cycle. The problem becomes easy to solve under the degraded chain promise.

Specifically, Bob will now know both X_i and Y_i for all $i \in Z$. If $j = 0$ or $j = q-1$, then Bob can already determine $\{i \mid X_i = Y_i = 0\}$, and can locally compute the final result. If $j \neq 0$ and $j \neq q-1$, then for any index $i' \notin Z$, Bob knows that $X_{i'} \neq j$. Thus if $Y_{i'} = j+1$, then $X_{i'}$ must be $j+1$ as well. This observation enables the following trick. Alice locally calculates $h_A = |\{i' \mid i' \notin Z \text{ and } j+1 \leq X_{i'} \leq q-1\}|$ and sends h_A to Bob, using $\log n$ bits. Bob calculates $h_B = |\{i' \mid i' \notin Z \text{ and } (j+1 \leq Y_{i'} \leq q-1 \text{ or } Y_{i'} = 0)\}|$. Given that $X_{i'} \neq j$ for $i' \notin Z$, one can easily verify from the cycle promise that $h_B - h_A$ is exactly the total number of indices i where $X_i = Y_i = 0$. The result for UNIONSIZECP(X, Y) is thus $n - (h_B - h_A)$. The amount of communication incurred in the above protocol is at most $O(\frac{n}{q}\log n + \log q + \log n) = O(\frac{n}{q}\log n + \log q)$ bits.

5.3 (ϵ, δ)-Approximate Communication Complexity

This section will prove following lower bound on the (ϵ, δ)-approximate communication complexity of UnionSizeCP:

Theorem 5.3.1. $\mathcal{R}_{\epsilon,\frac{1}{5}}(\text{UnionSizeCP}_{n,q}) = \Omega(\frac{1}{\epsilon q^2}) - O(\log \frac{1}{\epsilon})$ *for* $\epsilon = \Omega(\frac{1}{\sqrt{n}})$.

To obtain this lower bound, we will reduce from a new DisjointnessCP$_{n,q}$ problem, which is the natural extension of the standard Disjointness problem in binary strings [50]. We are interested in DisjointnessCP since its lower bound can be obtained via information cost arguments [4]. For discussion in this section, it will be convenient to consider an alternative form of the cycle promise (Figure 5.1). We will describe this promise and show that it is equivalent to the original one in Section 5.1. The reduction from Disjointness to UnionSizeCP are constructed in Section 5.3.1. Section 5.3.2 will lower bound Disjointness's communication complexity. Finally, we prove Theorem 5.3.1 in Section 5.3.3.

5.3.1 Reduction from DisjointnessCP

We define DisjointnessCP more formally as following:

Definition 5.3.1 (DisjointnessCP)**.** *In the* DisjointnessCP$_{n,q}$ *problem, Alice and Bob respectively hold X and Y, which are two strings of length n satisfying* $(X, Y) \in \mathcal{L}_q^n$ *where*

$$\mathcal{L}_q^n = \{(X, Y) \mid X \in \mathbb{Z}_q^n \text{ and } Y \in \mathbb{Z}_q^n \text{ and } (Y - X) \in \{0, 1\}^n\}.$$

The goal is to compute the function DisjointnessCP$_{n,q} : \mathcal{L}_q^n \to \{0, 1\}$ *defined as*

$$\text{DisjointnessCP}_{n,q}(X, Y) = \begin{cases} 0 & \exists i \in \{1, 2, ..., n\} \text{ such that } X_i = Y_i = 0 \\ 1 & \text{otherwise} \end{cases}$$

This section proves following theorem via a reduction from DisjointnessCP to UnionSizeCP:

Theorem 5.3.2. $\mathcal{R}_{\epsilon,\frac{1}{5}}(\text{UnionSizeCP}_{n,q}) \geq \mathcal{R}_{0,\frac{1}{5}}(\text{DisjointnessCP}_{\frac{1}{2\epsilon},q}) - 1$ *for* $\epsilon \geq \frac{1}{\sqrt{2n}}$.

Proof. Consider any given protocol for $\textsc{UnionSizeCP}_{n,q}$. Given a length-$\frac{1}{2\epsilon}$ input X for $\textsc{DisjointnessCP}_{\frac{1}{2\epsilon},q}$, Alice locally generates a length-n input X' by first replicating each character in X for $\frac{1}{\epsilon}$ times, and then appending 0 until the length of X' reaches n. This is always possible since $\frac{1}{2\epsilon^2} \leq n$. Bob generates Y' in a similar way. We now have:

- If $\textsc{DisjointnessCP}_{\frac{1}{2\epsilon},q}(X, Y) = 1$, then $\textsc{UnionSizeCP}_{n,q}(X', Y') = \frac{1}{2\epsilon^2}$.
- If $\textsc{DisjointnessCP}_{\frac{1}{2\epsilon},q}(X, Y) = 0$, then $\textsc{UnionSizeCP}_{n,q}(X', Y') \leq \frac{1}{2\epsilon^2} - \frac{1}{\epsilon}$.

One can easily verify that for all $\epsilon > 0$, we have $(1+\epsilon)(\frac{1}{2\epsilon^2} - \frac{1}{\epsilon}) < (1-\epsilon)\frac{1}{2\epsilon^2}$. Alice can now pick any value between $(1+\epsilon)(\frac{1}{2\epsilon^2} - \frac{1}{\epsilon})$ and $(1-\epsilon)\frac{1}{2\epsilon^2}$ as the threshold. Alice outputs 1 for $\textsc{DisjointnessCP}_{\frac{1}{2\epsilon},q}(X, Y)$ iff $\textsc{UnionSizeCP}_{n,q}(X', Y')$ returns a value above that threshold. Finally, Alice sends Bob a single bit to inform Bob of the result. We thus have:

$$\mathcal{R}_{\epsilon,\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}) \geq \mathcal{R}_{0,\frac{1}{5}}(\textsc{DisjointnessCP}_{\frac{1}{2\epsilon},q}) - 1$$

□

5.3.2 Communication Complexity of DisjointnessCP

This subsection lower bounds the communication complexity of DisjointnessCP as following theorem:

Theorem 5.3.3.

$$\begin{aligned}\mathcal{R}_0(\textsc{DisjointnessCP}_{n,q}) &= \Omega(\frac{n}{q^2}) - O(\log n)\\ \mathcal{R}_{0,\frac{1}{5}}(\textsc{DisjointnessCP}_{n,q}) &= \Omega(\frac{n}{q^2}) - O(\log n)\end{aligned}$$

Our proof for Theorem 5.3.3 will be almost entirely based on the information theoretic approach from [4]. In this approach, the *information complexity* of a function is used to lower bound the communication complexity of that function. Under certain conditions, it is further shown that the *conditional information complexity* of a

function is a lower bound on the function's information complexity. Next, under certain conditions, the approach establishes a direct-sum result between the conditional information complexity of a function (e.g., $\text{DISJOINTNESSCP}_{n,q}$) and the conditional information complexity of its constituent primitive function (e.g., $\text{DISJOINTNESSCP}_{1,q}$). Finally, the approach also provides some tools for reasoning about the conditional information complexity of such constituent primitive functions. The final lower bound on communication complexity obtained via this approach is for private-coin randomized protocols only. Since we will need a lower bound for public-coin protocols, at the end of this section, we will apply the well-known result from Newman [55] to convert this lower bound to a public-coin setting. Recall from Chapter 2 that the notation $\mathcal{R}_{0,\delta}$ simply means $\mathcal{R}_{\epsilon,\delta}$ with $\epsilon = 0$. We define $\mathcal{R}^{\text{pri}}_{0,\delta}(\text{DISJOINTNESSCP}_{n,q})$ to be the same as $\mathcal{R}_{0,\delta}(\text{DISJOINTNESSCP}_{n,q})$, except that $\mathcal{R}^{\text{pri}}_{0,\delta}$ is for private-coin protocols.

In the next, we first summarize the definitions and lemmas that we will use in this information theoretic approach. All these definitions (Definition 5.3.2 to 5.3.6) and lemmas (Lemma 5.3.1 to 5.3.4) are directly adapted from [4], and are not our contribution. See [4] for a more detailed discussion.

Definition 5.3.2 (Decomposable functions)**.** (Adapted from [4]) *If there are functions* $h : \mathcal{L}^1_q \to \{0,1\}$ *and* $g : \{0,1\}^n \to \{0,1\}$ *such that* $\text{DISJOINTNESSCP}_{n,q}(X,Y) = g(h(X_1,Y_1), h(X_2,Y_2), ..., h(X_n,Y_n))$*, then we say that* $\text{DISJOINTNESSCP}_{n,q}$ *is* g-decomposable with primitive h*. When the context is clear, we simply say that* $\text{DISJOINTNESSCP}_{n,q}$ *is* decomposable with primitive h*.*

We construct g as $g(x_1, x_2, ..., x_n) = \Pi_{i=1}^n x_i$. Then according to above definition, $\text{DISJOINTNESSCP}_{n,q}$ is a decomposable function with primitive $h = \text{DISJOINTNESSCP}_{1,q}$. For convenience, from now on in this section, h stands for the function $\text{DISJOINTNESSCP}_{1,q}$. Namely $h(x,y) = 0$ if $x = y = 0$, otherwise $h(x,y) = 1$.

Definition 5.3.3 (Mixture of product distributions)**.** (Adapted from [4]) *For random variables* X_i*,* Y_i*, and* T_i *(*$1 \le i \le n$*), their joint distribution* (X_i, Y_i, T_i) *is called a* mixture of product distribution *if conditioned on* T_i*,* X_i *and* Y_i *are independent.*

Let $\mathcal{T} = \{0,1\} \times \{1,2,..,q-1\}$, and let T_i $(1 \le i \le n)$ be a random variable drawn uniformly randomly from $\mathcal{T}$. Let X_i and Y_i $(1 \le i \le n)$ be two random variables depending on T_i where:

- If $T_i = (0, j)$, then $X_i = j$ and Y_i is drawn uniformly randomly from $\{j, j+1\}$.
- If $T_i = (1, j)$, then $Y_i = j$ and X_i is drawn uniformly randomly from $\{j, j-1\}$.

All additions and subtractions in the above are on $\mathbb{Z}_q$. Note that under this construction, it is impossible for $X_i = Y_i = 0$, which is intentional. Define ζ as the joint distribution of the above (X_i, Y_i, T_i). Clearly, conditioned on T_i, X_i and Y_i are independent. Hence ζ is a mixture of product distribution for all i's ($1 \le i \le n$).

Definition 5.3.4 (Collapsing distribution)**.** (Adapted from [4]) *A distribution on $\mathcal{L}_q^n$ is called a* collapsing distribution *for* DisjointnessCP$_{n,q}$ *with respect to h, if* DisjointnessCP$_{n,q}$ *is g-decomposable with primitive h, and if for all (X, Y)'s in the support of that distribution, all j's where $1 \le j \le n$, and all $(x, y) \in \mathcal{L}_q^1$, the following holds:*

$$g(h(X_1, Y_1), ..., h(X_{j-1}, Y_{j-1}), h(x, y), h(X_{j+1}, Y_{j+1}), ..., h(X_n, Y_n)) = h(x, y)$$

Define $\eta = \zeta^n$, and let $(X, Y, T) \sim \eta$. Consider the marginal distribution η_{XY} of (X, Y) in η. Since $(X_i, Y_i, T_i) \sim \zeta$, X_i and Y_i cannot simultaneously be 0, which means $h(X_i, Y_i) = 1$. Hence for all (X, Y)'s in the support of η_{XY}, we have $h(X_i, Y_i) = 1$ for all i's. This implies that $g(..., h(x, y), ...) = h(x, y)$, and thus η_{XY} is a collapsing distribution for DisjointnessCP$_{n,q}$.

Definition 5.3.5 (Conditional information cost)**.** (Adapted from [4]) *Let $\mathcal{P}$ be any two-party private-coin randomized protocol for* DisjointnessCP$_{1,q}$*. Let $(X_i, Y_i, T_i) \sim \zeta$, which is a mixture of product distributions on $\mathcal{L}_q^1 \times \mathcal{T}$. Given X_i and Y_i as the input to $\mathcal{P}$, the transmitted messages in $\mathcal{P}$ can be viewed as a random variable $\mathcal{P}(X_i, Y_i)$. The* conditional information cost *of $\mathcal{P}$ with respect to ζ (denoted as $\mathrm{CIC}_\zeta(\mathcal{P})$) is the mutual information between (X_i, Y_i) and $\mathcal{P}(X_i, Y_i)$ conditioned on T_i. Or formally:*

$$\mathrm{CIC}_\zeta(\mathcal{P}) = \sum_{t \in \mathcal{T}} I(\{(X_i, Y_i); \mathcal{P}(X_i, Y_i)\} | T_i = t) \Pr[T_i = t].$$

Here I stands for the standard notion of conditional mutual information [4].

Definition 5.3.6 (Conditional information complexity)**.** (Adapted from [4]) *Let $\mathcal{P}$ be any two-party private-coin randomized protocol for* DisjointnessCP$_{1,q}$*, such that for*

any input (x, y)*,* $\mathcal{P}$ *can generate the correct result with probability at least* $1 - \delta$*. The* δ-error conditional information complexity of $\textsc{DisjointnessCP}_{1,q}$ with respect to ζ*, denoted as* $\mathrm{CIC}_{\zeta,\delta}(\textsc{DisjointnessCP}_{1,q})$*, is defined as the minimum conditional information cost across all possible* $\mathcal{P}$*'s satisfying the earlier property.*

Lemma 5.3.1. (Adapted from [4]) *Consider* $\textsc{DisjointnessCP}_{n,q}$*, and the distribution* ζ*,* η*, and* η_{XY} *as defined earlier. We already know* $\textsc{DisjointnessCP}_{n,q}$ *is a decomposable function with primitive* $\textsc{DisjointnessCP}_{1,q}$*,* ζ *is a mixture of product distribution on* $\mathcal{L}_q^1 \times \mathcal{T}$*, and* η_{XY} *is a collapsing distribution for* $\textsc{DisjointnessCP}_{n,q}$ *with respect to* $\textsc{DisjointnessCP}_{1,q}$*. We must have:*

$$\mathcal{R}_{0,\delta}^{\mathrm{pri}}(\textsc{DisjointnessCP}_{n,q}) \geq n \times \mathrm{CIC}_{\zeta,\delta}(\textsc{DisjointnessCP}_{1,q}).$$

Lemma 5.3.2. (Adapted from [4]) *Let* Z *be a random variable uniformly randomly distributed on* $\{z_1, z_2\}$*, and let* $\Phi(z_1)$ *and* $\Phi(z_2)$ *be two additional random variables. If* $\Phi(z_1)$ *and* $\Phi(z_2)$ *are both independent of* Z*, then we have:*

$$I(Z; \Phi(Z)) \geq H^2(\Phi_{z_1}, \Phi_{z_2}).$$

Here Φ_{z_i} *is the distribution of* $\Phi(z_i)$*, and* H *is the Hellinger distance [4] between two distributions.*

Lemma 5.3.3. (Adapted from [4]) *For any two-party private-coin randomized protocol* $\mathcal{P}$*, let random variable* $\mathcal{P}(x, y)$ *denote the transmitted message in* $\mathcal{P}$ *under input* x *and* y*. Let* $\mathcal{P}_{x,y}$ *denote the distribution of* $\mathcal{P}(x, y)$*. For all* x*,* x'*,* y*, and* y'*, we have:*

$$2H^2(\mathcal{P}_{x,y}, \mathcal{P}_{x',y'}) \geq H^2(\mathcal{P}_{x,y}, \mathcal{P}_{x',y}) + H^2(\mathcal{P}_{x,y'}, \mathcal{P}_{x',y'}).$$

Lemma 5.3.4. (Adapted from [4]) *Let* $\mathcal{P}$ *be any private-coin randomized protocol for* $\textsc{DisjointnessCP}_{1,q}$*, such that for any input* (x, y)*,* $\mathcal{P}$ *can generate the correct result with probability at least* $1 - \delta$*. For any two input pairs* $(x, y) \in \mathcal{L}_q^1$ *and* $(x', y') \in \mathcal{L}_q^1$ *where* $\textsc{DisjointnessCP}_{1,q}(x, y) \neq \textsc{DisjointnessCP}_{1,q}(x', y')$*, we have:*

$$H^2(\mathcal{P}_{x,y}, \mathcal{P}_{x',y'}) \geq 1 - 2\sqrt{\delta}.$$

Having introduced the definitions and lemmas needed for the information theoretic arguments, we are now ready to prove Theorem 5.3.3.

Theorem 5.3.4. $\mathcal{R}^{\text{pri}}_{0,\delta}(\textsc{DisjointnessCP}_{n,q}) = \Omega(\frac{n}{q^2})$ *for any positive constant* $\delta \leq 0.22$.

Proof. Let $\mathcal{P}$ denote the optimal protocol with the minimum conditional information cost, across all possible two-party private-coin randomized protocols for $\textsc{DisjointnessCP}_{1,q}$ where for any input (x, y), the protocol can always generate the correct result with probability at least $1 - \delta$.

By Lemma 5.3.1 and Definition 5.3.5 and 5.3.6, we have:

$$\begin{aligned}
& \mathcal{R}^{\text{pri}}_{0,\delta}(\textsc{DisjointnessCP}_{n,q}) \\
\geq\ & n \times \text{CIC}_{\zeta,\delta}(\textsc{DisjointnessCP}_{1,q}) \\
=\ & n \times \text{CIC}_{\zeta}(\mathcal{P}) \\
=\ & \frac{n}{2(q-1)} \sum_{t \in \mathcal{T}} I(\{X_1, Y_1; \mathcal{P}(X_1, Y_1)\} \mid T_1 = t) \\
=\ & \frac{n}{2(q-1)} \sum_{j=1}^{q-1} (I(\{X_1, Y_1; \mathcal{P}(X_1, Y_1)\} \mid T_1 = (0, j)) + I(\{X_1, Y_1; \mathcal{P}(X_1, Y_1)\} \mid T_1 = (1, j)))
\end{aligned}$$

Conditioned on $T_1 = (0, j)$, (X_1, Y_1) is uniformly distributed on $\{(j, j), (j, j+1)\}$. Let $z_1 = (j, j)$, $z_2 = (j, j+1)$, and $Z = (X_1, Y_1)$. Lemma 5.3.2 tells us:

$$I(\{X_1, Y_1; \mathcal{P}(X_1, Y_1)\} \mid T_1 = (0, j)) \geq H^2(\mathcal{P}_{j,j}, \mathcal{P}_{j,j+1})$$

Similarly, we have:

$$I(\{X_1, Y_1; \mathcal{P}(X_1, Y_1)\} \mid T_1 = (1, j)) \geq H^2(\mathcal{P}_{j,j}, \mathcal{P}_{j-1,j})$$

Apply Cauchy inequality and triangle inequality, and we have:

$$\begin{aligned}
& \mathcal{R}^{\text{pri}}_{0,\delta}(\text{DISJOINTNESSCP}_{n,q}) \\
\geq\ & \frac{n}{2(q-1)} \sum_{j=1}^{q-1} (H^2(\mathcal{P}_{j,j}, \mathcal{P}_{j,j+1}) + H^2(\mathcal{P}_{j,j}, \mathcal{P}_{j-1,j})) \\
\geq\ & \frac{n}{4(q-1)^2} \left(\sum_{j=1}^{q-1} (H(\mathcal{P}_{j,j}, \mathcal{P}_{j,j+1}) + H(\mathcal{P}_{j,j}, \mathcal{P}_{j-1,j})) \right)^2 \\
\geq\ & \frac{n}{4(q-1)^2} \left(\sum_{j=1}^{q-1} H(\mathcal{P}_{j,j+1}, \mathcal{P}_{j-1,j}) \right)^2 \\
=\ & \frac{n}{4(q-1)^2} (H(\mathcal{P}_{1,2}, \mathcal{P}_{0,1}) + H(\mathcal{P}_{2,3}, \mathcal{P}_{1,2}) + \ldots + H(\mathcal{P}_{q-1,0}, \mathcal{P}_{q-2,q-1}))^2 \\
\geq\ & \frac{n}{4(q-1)^2} H^2(\mathcal{P}_{q-1,0}, \mathcal{P}_{0,1})
\end{aligned}$$

Next apply Lemma 5.3.3, and we have:

$$\begin{aligned}
\mathcal{R}^{\text{pri}}_{0,\delta}(\text{DISJOINTNESSCP}_{n,q}) &\geq \frac{n}{8(q-1)^2} H^2(\mathcal{P}_{q-1,0}, \mathcal{P}_{0,0}) + \frac{n}{8(q-1)^2} H^2(\mathcal{P}_{q-1,1}, \mathcal{P}_{0,1}) \\
&\geq \frac{n}{8(q-1)^2} H^2(\mathcal{P}_{q-1,0}, \mathcal{P}_{0,0})
\end{aligned}$$

Finally, apply Lemma 5.3.4, and we have:

$$\mathcal{R}^{\text{pri}}_{0,\delta}(\text{DISJOINTNESSCP}_{n,q}) \geq \frac{n}{8(q-1)^2}(1 - 2\sqrt{\delta}) = \Omega\left(\frac{n}{q^2}\right)$$

□

We can now prove Theorem 5.3.3:

Theorem 5.3.3 (Restated)**.**

$$\begin{aligned}
\mathcal{R}_0(\text{DISJOINTNESSCP}_{n,q}) &= \Omega(\frac{n}{q^2}) - O(\log n) \\
\mathcal{R}_{0,\frac{1}{5}}(\text{DISJOINTNESSCP}_{n,q}) &= \Omega(\frac{n}{q^2}) - O(\log n)
\end{aligned}$$

Proof. According to Newman [55],[2] we have:

$$\mathcal{R}^{\text{pri}}_{0,0.22}(\textsc{DisjointnessCP}_{n,q}) \leq \mathcal{R}_{0,0.2}(\textsc{DisjointnessCP}_{n,q}) + O(\log n + \log\log q)$$

Apply Theorem 5.3.4 and we have:

$$\mathcal{R}_{0,0.2}(\textsc{DisjointnessCP}_{n,q}) = \Omega\left(\frac{n}{q^2}\right) - O\left(\log n + \log\log q\right)$$

This lower bound is only non-trivial when $q < \sqrt{\frac{n}{\log n}}$. Thus we can discard the $\log\log q$ term for clarity:

$$\mathcal{R}_{0,0.2}(\textsc{DisjointnessCP}_{n,q}) = \Omega\left(\frac{n}{q^2}\right) - O\left(\log n\right)$$

Finally, apply Lemma 2.6.1 and we have $\mathcal{R}_0(\textsc{DisjointnessCP}_{n,q}) = \Omega\left(\frac{n}{q^2}\right) - O\left(\log n\right)$ as well. □

5.3.3 Proof for Theorem 5.3.1

Theorem 5.3.1 (Restated). $\mathcal{R}_{\epsilon,\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}) = \Omega(\frac{1}{\epsilon q^2}) - O(\log\frac{1}{\epsilon})$ *for* $\epsilon = \Omega(\frac{1}{\sqrt{n}})$.

Proof. For $\epsilon \geq \frac{1}{\sqrt{2n}}$, directly combining Theorem 5.3.2 and Theorem 5.3.3 yields the proof. We still need to cover the case for $\epsilon = \Omega(\frac{1}{\sqrt{n}})$ but $\epsilon < \frac{1}{\sqrt{2n}}$. For such ϵ (which is necessarily $\Theta(\frac{1}{\sqrt{n}})$), we have:

$$\begin{aligned}
\mathcal{R}_{\epsilon,\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}) &\geq \mathcal{R}_{\frac{1}{\sqrt{2n}},\frac{1}{5}}(\textsc{UnionSizeCP}_{n,q}) \\
&= \Omega(\frac{\sqrt{n}}{q^2}) - O(\log n) \\
&= \Omega(\frac{1}{\epsilon q^2}) - O(\log\frac{1}{\epsilon})
\end{aligned}$$

□

[2] Newman's original result was only stated for functions, while here we are dealing with the partial functions of DisjointnessCP. Nevertheless, Newman's original proof actually holds without modification to partial functions.

Chapter 6

The Fundamental Roles of Cycle Promise and UnionSizeCP

Our reduction from UnionSizeCP so far has led to the exponential gap result for Sum, when $b \leq N^{1-c}$ or $\frac{1}{\epsilon^{0.5-c}}$ for any positive constant $c < 0.25$. This restriction on b comes from the $\frac{1}{q^2}$ term in the lower bound of the communication complexity of UnionSizeCP. Our upper bound on UnionSizeCP indicates that such a polynomial dependency on $\frac{1}{q}$ is unavoidable because of the cycle promise. It is thus natural to ask: Can we reduce from problems without promises? Or can we reduce from problems with a different promise, to weaken the polynomial dependency on $\frac{1}{q}$ to $\log \frac{1}{q}$? For *any* possible *oblivious reduction* (defined next) from *any* two-party communication complexity problem Π to Sum, this section answers these questions in the negative. Specifically, we will prove the *completeness* of UnionSizeCP in the sense that such a Π can always be reduced to UnionSizeCP and must have a communication complexity no larger than that of $\text{UnionSizeCP}_{N,\lfloor\sqrt{b/3}\rfloor}$. Thus any FT lower bound on Sum, obtained in such a way via Π, must contain some polynomial term of $\frac{1}{b}$. Overcoming this polynomial term in the lower bound might still be possible, but one would have to resort to methods other than oblivious reductions from two-party problems. Our proof also (implicitly) shows that the cycle promise can be derived and that the promise likely plays a fundamental role in reasoning about many functions beyond Sum.

In the following, Section 6.1 defines the notion of oblivious reductions. Section 6.2

presents the completeness theorem and its proof overview. Finally, Section 6.3 proves the theorem.

6.1 Oblivious Reductions

Consider any two-party communication complexity problem Π, where (only) Alice aims to learn $\Pi(X, Y)$. In a (general) reduction from Π to SUM, Alice and Bob are given some black-box *oracle* fault-tolerant protocol for SUM, and they are supposed to use this oracle to solve Π with any given input pair (X, Y). Since the (global) oracle protocol is distributed, it will be convenient to imagine that each node in the topology has its own oracle protocol, and invoking these protocols in a "consistent" fashion will enable the root to produce a meaningful result.

In an *oblivious reduction* to SUM, there is some fixed topology G and for each (X, Y) pair, there exists some *reference setting* specifying the value and failure time of each node in G. The reference settings are oblivious to the oracle. As explained in Chapter 4, a reference setting here should not fail or disconnect nodes with a value of 1. The zero-error SUM result in the reference setting should be the same as $\Pi(X, Y)$, so we can directly use it for solving Π. The reduction protocol is required to be oblivious as well. Specifically, Alice and Bob first pick a (public) random string. Next before invoking the oracle and purely based on X (Y), Alice (Bob) decides for each node in G, exactly up to which round she (he) will invoke the oracle. Note that to invoke the oracle for a certain round, Alice/Bob needs to invoke the oracle for all previous rounds as well. Alice (Bob) also decides the (initial) value of each node for which she (he) will invoke the oracle for at least one round. Requiring Alice and Bob to make these decisions beforehand is the most important aspect of oblivious reductions. We define the *reference execution* for (X, Y) to be the (global) oracle's execution under the reference setting for (X, Y) and under the chosen random string. To enable the root to generate a meaningful result, we require that the initial value, incoming messages, and coin flips fed by Alice/Bob into the oracle protocol on a node be the same as those fed into that node's oracle in the reference execution for (X, Y). Furthermore, after a node has failed in the reference execution, Alice/Bob must not invoke that node's oracle any more (since that node can no longer help out). Finally, there are two special nodes α and β in G, such that Alice and Bob will

always invoke the oracle on α and β (respectively) until the root generates a result. Here α must be the root of G,[1] while β can be any other node. During the reduction, Alice (Bob) may only send to the other party all those outgoing messages generated by the oracle invocation on node α (β). This allows the establishment of a simple factor-2 relation between the communication complexity of Π and SUM.

Our previous reduction from UNIONSIZECP to SUM are oblivious reductions. Besides those two specific instances, the broad class of oblivious reductions further captures reductions from *any* two-party problem Π with *any* promise, using *any* topology G with *any* proper reference settings.

6.2 The Completeness of UNIONSIZECP

We now present a strong result on the completeness of UNIONSIZECP, in oblivious reductions:

Theorem 6.2.1. *Consider any two-party communication complexity problem* Π *that can be obliviously reduced to* SUM *for some topology* G *with* N *nodes, with the* SUM *oracle protocol having a time complexity of up to* b *flooding rounds where* $b \geq 12$*. For all* $t \geq 1$*, we have:*

$$\begin{aligned} \mathcal{R}_0^{\text{syn}}(\Pi, t) &\leq \mathcal{R}_0^{\text{syn}}(\text{UNIONSIZECP}_{N,\lfloor\sqrt{b/3}\rfloor}, t) \\ \mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\Pi, t) &\leq \mathcal{R}_{\epsilon,\delta}^{\text{syn}}(\text{UNIONSIZECP}_{N,\lfloor\sqrt{b/3}\rfloor}, t) \end{aligned}$$

The following gives an overview of the proof of this theorem, and also defines some useful concepts. Let $\mathcal{X}$ be Alice's input domain in Π, and $\mathcal{Y}$ be Bob's. Let $\mathcal{L} \subseteq \mathcal{X} \times \mathcal{Y}$ be the set of all valid input pairs, given the promise in Π. If Π has no promise, then $\mathcal{L} = \mathcal{X} \times \mathcal{Y}$. Given $(X, Y) \in \mathcal{L}$, an oblivious reduction has a reference setting specifying the value of each node in G. For any node τ where $\tau \neq \alpha$ and $\tau \neq \beta$, we define τ's *(value) assignment graph* to be the bipartite graph where $\mathcal{X} \cup \mathcal{Y}$ are vertices and an edge (X, Y) exists iff $(X, Y) \in \mathcal{L}$. In addition, each edge (X, Y) has a binary label which is the value of τ in the reference setting for (X, Y).

[1] This is largely for clarity, and can be relaxed if desired.

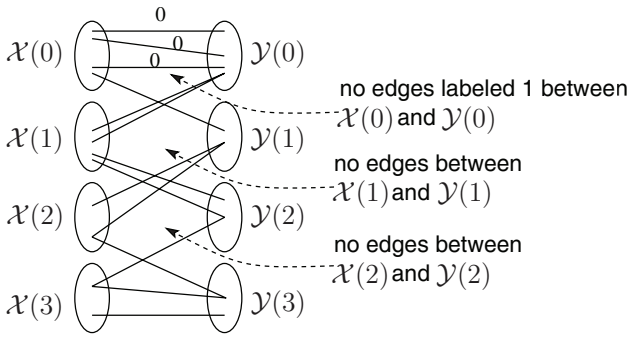

Figure 6.1: Example assignment graph for a given node τ and for $b' = 4$. $\mathcal{X}(0)$, $\mathcal{Y}(0)$, ..., $\mathcal{X}(3)$, and $\mathcal{Y}(3)$ are the 8 subsets, which may have different sizes and different numbers of incidental edges. All edges without labels indicated have a label of 1.

We will prove that it is always possible to partition the vertices in τ's assignment graph into $2b'$ (where $b' = \lfloor \sqrt{b/3} \rfloor \geq 2$) disjoint subsets with strong properties as illustrated in Figure 6.1. Intuitively, this is because otherwise the reference setting for some input pair would need to have so many failures in G such that τ (with a value of 1) would be disconnected from the root. Those failures are needed to ensure that Alice (Bob) can invoke the oracle on α (β) throughout the execution.

At this point, we already have something close to the cycle promise — if we view each subset as a super vertex, then all the $2b'$ super vertices form a subgraph of a length-$2b'$ cycle. It is now possible to reduce Π to UNIONSIZECP$_{N,b'}$, by mapping an input X for Π to an input X' for UNIONSIZECP as following: Each τ in G corresponds to a unique i ($1 \leq i \leq N-2$), and X'_i is set to be the index of the subset in τ's assignment graph to which X belongs. Finally, X'_{N-1} is set to be the (initial) value of α in the given oblivious reduction, which can be obtained purely based on X. X'_N is set to be 0. The conversion from Y to Y' is similar, with $Y'_{N-1} = 0$ and Y'_N being the value of β.

6.3 Proof for Theorem 6.2.1

We first formalize the notion of a node τ being disconnected from the root in a simulation. Consider any input pair $(X, Y) \in \mathcal{L}$ and the corresponding reference

setting in the oblivious reduction from Π to SUM. Define $\Phi(X, Y)$ to be the execution of the SUM oracle protocol under that reference setting and under the (public) random string chosen by Alice and Bob in the oblivious reduction. For a given node τ in G where $\tau \neq \alpha$ and $\tau \neq \beta$, we say that τ is *disconnected* form the root in an execution $\Phi(X, Y)$, if either τ fails during $\Phi(X, Y)$, or the root and τ are no longer in the same connected component at the end of $\Phi(X, Y)$. We have the following trivial lemma:

Lemma 6.3.1. *For any $(X, Y) \in \mathcal{L}$, if τ has a value of 1 under the reference setting for (X, Y), then in $\Phi(X, Y)$, τ is never disconnected from the root.*

Proof. Trivially follows from the requirement on the reference settings in oblivious reductions. □

Next consider the assignment graph of any node τ where $\tau \neq \alpha$ and $\tau \neq \beta$. Let $b' = \lfloor \sqrt{b/3} \rfloor$, which implies $b' \geq 2$ since $b \geq 12$. We partition the vertices of τ's assignment graph into $2b'$ disjoint subsets of $\mathcal{X}(0), \mathcal{Y}(0), \ldots, \mathcal{X}(b'-1)$, and $\mathcal{Y}(b'-1)$ in the following way. For all integer $i \in [0, b'-3]$, we recursively define:

$$
\begin{aligned}
\mathcal{X}(0) &= \{X \mid \tau(X, Y) = 0 \text{ for some } (X, Y) \in \mathcal{L}\} \\
\mathcal{Y}(0) &= \{Y \mid \tau(X, Y) = 0 \text{ for some } (X, Y) \in \mathcal{L}\} \\
\mathcal{X}(i+1) &= (\mathcal{X} \setminus \cup_{j=0}^{i} \mathcal{X}(j)) \cap \{X \mid \tau(X, Y) = 1 \text{ for some } (X, Y) \in \mathcal{L} \text{ where } Y \in \mathcal{Y}(i)\} \\
\mathcal{Y}(i+1) &= (\mathcal{Y} \setminus \cup_{j=0}^{i} \mathcal{Y}(j)) \cap \{Y \mid \tau(X, Y) = 1 \text{ for some } (X, Y) \in \mathcal{L} \text{ where } X \in \mathcal{X}(i)\} \\
\mathcal{X}(b'-1) &= \mathcal{X} \setminus \cup_{j=0}^{b'-2} \mathcal{X}(j) \\
\mathcal{Y}(b'-1) &= \mathcal{Y} \setminus \cup_{j=0}^{b'-2} \mathcal{Y}(j)
\end{aligned}
$$

Figure 6.1 illustrated these sets for a given τ for $b' = 4$. Intuitively, any vertex with some 0-labeled incidental edge belongs to $\mathcal{X}(0)$ or $\mathcal{Y}(0)$. Thus an edge (X, Y) always has a label of 1 if $X \notin \mathcal{X}(0)$ or $Y \notin \mathcal{Y}(0)$. We say that there are edges between two sets $\mathcal{X}(i)$ and $\mathcal{Y}(j)$ iff there exists some edge (X, Y) in the assignment graph for some $X \in \mathcal{X}(i)$ and some $Y \in \mathcal{Y}(j)$.

Section 6.4 will prove the following key lemma regarding τ's assignment graph. The proof uses only elementary techniques but is still rather involved, due to the need to capture all possible topologies. Hence we leave the proof to a seperate section.

Lemma 6.3.2. *In any oblivious reduction from* Π *to* SUM*, consider any node* τ *in* G *where* $\tau \neq \alpha$ *and* $\tau \neq \beta$*. In* τ*'s assignment graph, there must be no edges labeled 1 between* $\mathcal{X}(0)$ *and* $\mathcal{Y}(0)$*, and no edges between* $\mathcal{X}(i)$ *and* $\mathcal{Y}(i)$ *for all* $1 \leq i \leq b' - 2$*.*

Note that it is still possible for edges to exist between $\mathcal{X}(b'-1)$ and $\mathcal{Y}(b'-1)$. Intuitively, this lemma holds because if there existed an edge (X, Y) satisfying the property described in the lemma, then the reference setting for (X, Y) would need to have so many failures in G such that τ would be disconnected from the root. Those failures are needed to ensure that Alice (Bob) can invoke the oracle on α (β) throughout the execution (i.e., to ensure that α and β remain unspoiled). On the other hand, τ must have a value of 1 in the reference setting for such (X, Y). This contradicts with Lemma 6.3.1.

Next we can use Lemma 6.3.2 to prove Theorem 6.2.1:

for Theorem 6.2.1. By the condition in the theorem, there exists some oblivious reduction $\mathcal{P}$ from Π to SUM for some topology G. Let the $N-2$ nodes other than α and β in G be $\tau_1, \tau_2, \ldots, \tau_{N-2}$. Consider any input pair $(X, Y) \in \mathcal{L}$. Let $\tau_i(X, Y)$, $\alpha(X, Y)$, and $\beta(X, Y)$ be the values of τ_i, α, and β in $\mathcal{P}$'s reference setting for (X, Y), respectively. Since the zero-error SUM result under the reference setting must be the same as $\Pi(X, Y)$ and since the reference setting never fails or disconnects nodes with a value of 1, we have:

$$\Pi(X, Y) \quad = \quad \alpha(X, Y) + \beta(X, Y) + \sum_{i=1}^{N-2} \tau_i(X, Y)$$

We intend to reduce Π to UNIONSIZECP$_{N,b'}$. To do so, Alice converts her input X for Π to a corresponding input X' of length N for UNIONSIZECP in the following way, using only local knowledge. Let X'_i be the ith character in the string X'. Alice sets the last character X'_N to be 0. Alice next leverages $\mathcal{P}$ to obtain the value of $\alpha(X, Y)$, without communicating with Bob. To do so, Alice invokes $\mathcal{P}$ using X and then stops once $\mathcal{P}$ needs to invoke the SUM oracle protocol (which Alice does not have). Doing so clearly does not incur any communication. By definition of oblivious reductions, $\mathcal{P}$ at this point must have decided, based on X, the (initial) value of node α. Furthermore, this value must be the same as $\alpha(X, Y)$. Alice now obtains $\alpha(X, Y)$ purely based

on local knowledge. Alice sets X'_{N-1} to this value. Next Alice needs to determine X'_i for $1 \leq i \leq N-2$. By definition of oblivious reductions, $\mathcal{P}$ needs to specify the reference setting corresponding to each input pair $(X, Y) \in \mathcal{L}$. Using such information in $\mathcal{P}$, Alice can determine the assignment graph of τ_i for $1 \leq i \leq N-2$. In τ_i's assignment graph, Alice's current input X must belong to exactly one of the subsets of vertices. Let $\mathcal{X}(j)$ be the subset to which X belongs. Alice then sets $X'_i = j$.

Bob constructs input Y' of length N for UNIONSIZECP similarly, using only his local knowledge of Y. Specifically, Bob sets $Y'_{N-1} = 0$ and $Y'_N = \beta(X, Y)$. Same as earlier, Bob can obtain $\beta(X, Y)$ via $\mathcal{P}$, without communicating with Alice. Next for each i where $1 \leq i \leq N-2$, Bob sets Y'_i to be j where $\mathcal{Y}(j)$ is the subset to which Y belongs to, in τ_i's assignment graph.

We next show that X' and Y' are strings satisfying the cycle promise with $q = b'$. First, for $i = N-1$ or N, obviously X'_i and Y'_i satisfy the cycle promise. Next consider any $i \in [1, N-2]$ and τ_i's assignment graph. Clearly X'_i and Y'_i are integer in $[0, b'-1]$ for all such i. By construction of the assignment graph, we know that there are

- no edges between $\mathcal{X}(0)$ and $\mathcal{Y}(j)$ for all $j \geq 2$,
- no edges between $\mathcal{X}(b'-1)$ and $\mathcal{Y}(j)$ for all $j \leq b'-3$, and
- no edges between $\mathcal{X}(j_1)$ and $\mathcal{Y}(j_2)$ for all $1 \leq j_1 \leq b'-2$ and $|j_1 - j_2| \geq 2$.

Furthermore, Lemma 6.3.2 shows that there are no edges between $\mathcal{X}(j)$ and $\mathcal{Y}(j)$ for all $1 \leq j \leq b'-2$. Since $(X, Y) \in \mathcal{L}$, there must exist an edge between X and Y in τ_i's assignment graph. Thus X'_i and Y'_i must satisfy the cycle promise.

Finally, consider any given protocol for UNIONSIZECP$_{N,b'}$. Alice and Bob invoke that protocol using X' and Y' as inputs, respectively. We claim that UNIONSIZECP(X', Y')

can be used directly as the result of $\Pi(X, Y)$, since:

$$
\begin{aligned}
& \textsc{UnionSizeCP}(X', Y') \\
= \; & |\{i \mid (1 \le i \le N) \text{ and } (X'_i \neq 0 \text{ or } Y'_i \neq 0)\}| \\
= \; & \alpha(X, Y) + \beta(X, Y) + |\{i \mid (1 \le i \le N-2) \text{ and } (X \notin \mathcal{X}(0) \text{ for } \tau_i \text{ or } Y \notin \mathcal{Y}(0) \text{ for } \tau_i)\}| \\
= \; & \alpha(X, Y) + \beta(X, Y) + |\{i \mid 1 \le i \le N-2 \text{ and } \tau_i(X, Y) = 1\}| \\
& \text{(by construction of the assignment graph and Lemma 6.3.2)} \\
= \; & \alpha(X, Y) + \beta(X, Y) + \sum_{i=1}^{N-2} \tau_i(X, Y) = \Pi(X, Y)
\end{aligned}
$$

In the above derivation, we have leveraged Lemma 6.3.2 which shows that there are no edges labeled 1 between $\mathcal{X}(0)$ and $\mathcal{Y}(0)$. Together with the construction of the assignment graph, this means that $\tau_i(X, Y) = 1$ if and only if in τ_i's assignment graph, $X \notin \mathcal{X}(0)$ or $Y \notin \mathcal{Y}(0)$. □

6.4 Proof for Lemma 6.3.2

Recall the various concepts defined in Sections 6.2 and 6.3.

Proof for Lemma 6.3.2. The lemma trivially holds for $b' = 2$, and thus we only need to consider $b' \ge 3$ (or equivalently $b \ge 27$). We prove this lemma via a contradiction and assume that the lemma does not hold. Intuitively, we will construct a path in τ's assignment graph (not a path in G) where vertices on the path are individual inputs. Since it is a path in the assignment graph, by the definition of τ's assignment graph, any two adjacent inputs on that path must form a valid input pair in $\mathcal{L}$. The path will start from some vertex in $\mathcal{X}(0)$ and end with some vertex in $\mathcal{Y}(0)$. Furthermore, all edges on the path have a label of 1, and the path has no more than $2(b'-1)-1$ hops. These properties can be later used to find a contradiction.

We now present the formal proof. If the lemma does not hold, then in the assignment graph of τ, either there is an edge labeled 1 between $\mathcal{X}(0)$ and $\mathcal{Y}(0)$, or there is an edge (which must have a label of 1) between $\mathcal{X}(j)$ and $\mathcal{Y}(j)$ for some $j \in [1, b'-2]$. We claim that in either case, we can find in the assignment graph a path $X^{(0)}$, $Y^{(1)}$, $X^{(1)}$, $Y^{(2)}$, ..., $X^{(k)}$, $Y^{(k+1)}$ for some $k \in [0, b'-2]$, such that:

- $X^{(i)} \in \mathcal{X}$ for $0 \leq i \leq k$, and $Y^{(i)} \in \mathcal{Y}$ for $1 \leq i \leq k+1$,
- $X^{(0)} \in \mathcal{X}(0)$ and $Y^{(k+1)} \in \mathcal{Y}(0)$, and
- all edges in the path have a label of 1.

If there is an edge labeled 1 between $\mathcal{X}(0)$ and $\mathcal{Y}(0)$, we simply set $k = 0$ and let $X^{(0)}$ and $Y^{(1)}$ be the two endpoints of that edge, respectively. Our claim then trivially holds. If there is an edge (X, Y) where $X \in \mathcal{X}(j)$ and $Y \in \mathcal{Y}(j)$ for some $j \in [1, b'-2]$, then we set $k = j$. First consider the case where k is even. By the construction of the assignment graph, X must be connected with some vertex in $\mathcal{Y}(j-1)$, and that vertex must be connected to some vertex in $\mathcal{X}(j-2)$, and so on. Since k is even, there must be some path (with exactly k hops) in the assignment graph from X to some $X^{(0)} \in \mathcal{X}(0)$. Similarly, there must be some path (with exactly k hops) in the assignment graph from Y to some $Y^{(k+1)} \in \mathcal{Y}(0)$. These two paths, together with the edge between X and Y, exactly form the path needed by our claim. The case for odd k is similar.

Given the above path $X^{(0)}$, $Y^{(1)}$, $X^{(1)}$, $Y^{(2)}$, ..., $X^{(k)}$, $Y^{(k+1)}$ (satisfying all the above properties), we define $\mathcal{I}$ (where $\mathcal{I} \subseteq \mathcal{L}$) to be the set of input pairs (X, Y) such that (X, Y) is an edge in that path. We also call $\mathcal{I}$ as the *problematic input set*. By construction of $\mathcal{I}$, we know that τ has a value of 1 in the reference setting for any $(X, Y) \in \mathcal{I}$. Lemma 6.4.1 next proves that for all $(X, Y) \in \mathcal{I}$, τ is disconnected from the root by the end of the execution of $\Phi(X, Y)$. This leads to a contradiction with Lemma 6.3.1. □

Lemma 6.4.1. *Suppose $b \geq 27$. Consider the problematic input set $\mathcal{I}$ as constructed in the proof of Lemma 6.3.2. For all $(X, Y) \in \mathcal{I}$, τ is disconnected from the root by the end of the execution of $\Phi(X, Y)$.*

The rest of this section will prove Lemma 6.4.1. This lemma is challenging to prove, and we need much preparation work in Sections 6.4.1 and 6.4.2 before actually proving it in Section 6.4.3.

6.4.1 Node α and β Must Remain Unspoiled

We start by proving that node α (β) must remain unspoiled for Alice (Bob) in an oblivious reduction. To do so, we inherit the formal framework developed in Section 4.3. Since for each input pair $(X, Y) \in \mathcal{L}$, there is a corresponding reference setting in the oblivious reduction, the notions of values and failure time of nodes in Section 4.3 are still well-defined here. Namely, all we need to do is to replace the notion of "simulated execution under (X, Y)" in Section 4.3 by the notion of "reference setting for (X, Y)".[2] All concepts defined in Section 4.3 now carry over directly without modification. For example, a node v is a value epicenter for Alice's input X if its value in the reference setting is not uniquely determined by X.

Lemma 4.3.1 in Section 4.3 proved that Alice can simulate all unspoiled nodes. In this section, we intend to prove the reverse for oblivious reductions — if a node is spoiled for Alice in a round r, then in an oblivious reduction, Alice can never invoke the oracle protocol on that node for round r. Since in an oblivious reduction Alice (Bob) is required to invoke the oracle on node α (β) throughout the execution, this in turn implies that α (β) must remain unspoiled for Alice (Bob). Our proof will hinge upon the property of oblivious reductions, which requires Alice (Bob) to decide, beforehand, exactly up to which rounds she (he) will invoke the oracle on each node.

Lemma 6.4.2. *Consider any oblivious reduction from* Π *to* SUM. *If a node v in G is spoiled for Alice's input X (Bob's input Y) in round $r' \geq 0$, then when Alice (Bob) has the input X (Y), Alice (Bob) will not invoke the oracle on v for round r'.*

Proof. We only need to prove the part for Alice. Let $r \in [1, r']$ be the very first round during which v is spoiled. It suffices to prove that Alice will not invoke the oracle on v for round r — since the oracle protocol carries internal state from round to round, Alice can never invoke the oracle for round r' without invoking the oracle for earlier rounds.

If round r is the first round during which v is spoiled, there must exist some epicenter u_0 with an occurrence time of r_0 ($r_0 \leq r$) such that there exists a spoil path from u_0

[2]These two notions are actually exactly the same. In Section 4.3 there was no need to introduce the more formal notion of reference settings, so there we used the notion of simulated execution.

to v with exactly $l = r - r_0$ hops. To show that Alice will not invoke the oracle on v for round r, we use an induction on l.

If $l = 0$, v itself must be an epicenter occurring at round r. We consider two cases. If v is a value epicenter, then the occurrence time is round 1 and $r = 1$. In an oblivious reduction, Alice needs to decide purely based on X, the input value of each node for which she will invoke the oracle for at least one round. This means that Alice must never invoke the oracle on v — otherwise she risks deviating from the corresponding execution under the reference setting. Next if v is a failure epicenter, then round r (i.e., the occurrence time of the epicenter) must be the earliest possible failure time. This means that there exists Bob's inputs Y and Y', such that v's failure time is exactly round r in the reference setting for (X, Y) and is after round r in the reference setting for (X, Y'). If Alice decides that she will invoke the oracle on v for round r, then again she risks deviating from the execution under the reference setting since the reference setting could be (X, Y).

For the inductive step, assume that the lemma holds for all values up to l and we consider $l + 1$. Again, there exists some epicenter u_0 with an occurrence time of r_0 ($r_0 \leq r$) such that there exists a spoil path from u_0 to v with exactly $l + 1$ hops. Consider the node u immediately before v in this spoil path. Then the length of the spoil path from u_0 to u is exactly l hops, and u is spoiled in round $r - 1$, where $r - 1 \geq 1$. By the inductive hypothesis, Alice (with an input X) does not invoke the oracle on u for round $r - 1$. Next we prove via a contradiction and assume that Alice still invokes the oracle on v for round r. Note that in an oblivious reduction, the only way for Alice to obtain the potential message sent in round $r - 1$ by the oracle protocol on u ($u \neq \beta$) is for Alice to invoke the oracle on u for round $r - 1$ herself. Thus for Alice to still invoke the oracle on v for round r, u must have failed in round $r - 1$ or earlier in all the reference settings for all possible input pairs (X, Y) given the current X. We claim that it is impossible for u to fail exactly in round $r - 1$ in all these reference settings, since otherwise this failure is a stable failure for X, and there would be no spoil path from u_0 to v via u. Thus there must exist some Y such that u fails before round $r - 1$. This in turn, means that the occurrence time of the epicenter u is round $r - 2$ or earlier. Thus v is spoiled by u in round $r - 1$ or earlier, which contradicts with the fact that r is the first round that v becomes spoiled. □

Corollary 6.4.1. *Consider any oblivious reduction from* Π *to* SUM. *For any input*

pair $(X, Y) \in \mathcal{L}$*,* α *(*β*) must remain unspoiled for Alice's input* X *(Bob's input* Y*) throughout the execution of* $\Phi(X, Y)$*.*

Proof. Trivially follows from Lemma 6.4.2 and the fact that in an oblivious reduction, Alice (Bob) is required to invoke the oracle on α (β) throughout the entire execution. □

The above corollary is all we need for the proofs next — we no longer need Lemma 6.4.2.

6.4.2 Reasoning about Paths – Some Technical Lemmas

This section proves a series of technical lemmas, which we will later use to prove Lemma 6.4.1. We start with some useful concepts and definitions, as summarized in Table 6.1.

Some useful concepts and definitions. Throughout this section, we use $\mathcal{I}$ to denote the problematic input set as constructed in the proof of Lemma 6.3.2.

A *path* p in the topology G is a sequence of nodes $(v_1, v_2, ..., v_k)$ such that $k \geq 2$ and for all $i \in [1, k-1]$, v_{i+1} is a neighbor of v_i in G. For any node v, $v \in p$ simply means that v appears in p. A path p is a *simple path* if no node appears more than once in the path. All paths we discuss will be simple paths. The *length* of a path p, denoted as $|p|$, is defined as the number of nodes in p minus 1. Consider any node τ in G, where $\tau \neq \alpha$ and $\tau \neq \beta$. With respect to τ, an α-path is a path from τ to α without passing β. Formally, it is a path $(v_1, v_2, ..., v_k)$ satisfying $v_1 = \tau$, $v_k = \alpha$, and $v_i \neq \beta$ for all $i \in [2, k-1]$. We similarly define β-paths with respect to τ, as paths from τ to β without passing α. We will only discuss α-paths and β-paths with respect to τ, and thus we will drop the phrase "with respect to τ". As we will easily prove later, since α is itself the root, any path p from τ to the root must contain an α-path or a β-path as a part.

For any α-path or β-path p, we say that p is *cut* in a certain round if some node (potentially τ) in p fails in or before that round. Given the problematic input set $\mathcal{I}$,

$\mathcal{I}$	the problematic input set as constructed in the proof of Lemma 6.3.2.
$\Phi(X, Y)$	the execution of the SUM oracle protocol under the reference setting for (X, Y) and under the given public coin outcomes chosen by Alice and Bob
$\lambda(X, Y)$	number of rounds in each flooding round in the execution $\Phi(X, Y)$
$F_A(X, v)$	v's failure time in the simulation, if v has a stable failure with respect to Alice's input X
$F_B(Y, v)$	v's failure time in the simulation, if v has a stable failure with respect to Bob's input Y
p	a simple path in the topology G
$\|p\|$	length of the path p
α-path	path from τ to α without passing β
β-path	path from τ to β without passing α
dummy path	a path that will be cut by the end of the execution of $\Phi(X, Y)$ for all $(X, Y) \in \mathcal{I}$.
$p_A(X, t)$ or $p(X, t)$	$\exists v \in p$ such that $F_A(X, v) \leq t\|p\|$
$p_B(Y, t)$ or $p(Y, t)$	$\exists v \in p$ such that $F_B(Y, v) \leq t\|p\|$
$\mathbb{P}^\alpha$	the finite set of all non-dummy α-paths
$\mathbb{P}^\beta$	the finite set of all non-dummy β-paths
$\mathbb{P}^\alpha_{<p}$	the set of all paths in $\mathbb{P}^\alpha$ whose lengths are smaller than the length of p
$\mathbb{P}^\beta_{<p}$	the set of all paths in $\mathbb{P}^\beta$ whose lengths are smaller than the length of p
$\mathbb{P}^\alpha(X, t)$	either $\mathbb{P}^\alpha = \emptyset$ or $p'(X, t)$ holds for all $p' \in \mathbb{P}^\alpha$
$\mathbb{P}^\alpha(Y, t)$	either $\mathbb{P}^\alpha = \emptyset$ or $p'(Y, t)$ holds for all $p' \in \mathbb{P}^\alpha$
$\mathbb{P}^\beta(X, t)$	either $\mathbb{P}^\beta = \emptyset$ or $p'(X, t)$ holds for all $p' \in \mathbb{P}^\beta$
$\mathbb{P}^\beta(Y, t)$	either $\mathbb{P}^\beta = \emptyset$ or $p'(Y, t)$ holds for all $p' \in \mathbb{P}^\beta$
$\mathbb{P}^\alpha_{<p}(X, t)$	either $\mathbb{P}^\alpha_{<p} = \emptyset$ or $p'(X, t)$ holds for all $p' \in \mathbb{P}^\alpha_{<p}$
$\mathbb{P}^\alpha_{<p}(Y, t)$	either $\mathbb{P}^\alpha_{<p} = \emptyset$ or $p'(Y, t)$ holds for all $p' \in \mathbb{P}^\alpha_{<p}$
$\mathbb{P}^\beta_{<p}(X, t)$	either $\mathbb{P}^\beta_{<p} = \emptyset$ or $p'(X, t)$ holds for all $p' \in \mathbb{P}^\beta_{<p}$
$\mathbb{P}^\beta_{<p}(Y, t)$	either $\mathbb{P}^\beta_{<p} = \emptyset$ or $p'(Y, t)$ holds for all $p' \in \mathbb{P}^\beta_{<p}$

Table 6.1: Notations and definitions used in Section 6.4.2 and 6.4.3

we say that an α-path or β-path p is *dummy* if for all $(X, Y) \in \mathcal{I}$, p is cut by the end of the execution $\Phi(X, Y)$. Otherwise p is *non-dummy*. A non-dummy path p may still be cut in the execution of $\Phi(X, Y)$ for some $(X, Y) \in \mathcal{I}$. Intuitively, a dummy path p can be easily dismissed in our proofs later since we will be focusing on the input pairs in $\mathcal{I}$ and a dummy path is always cut in the corresponding executions. So usually we will only need to focus on non-dummy paths. We use $\mathbb{P}^\alpha$ ($\mathbb{P}^\beta$) to denote the finite set of all non-dummy α-paths (β-paths). Note that the paths in $\mathbb{P}^\alpha$ and $\mathbb{P}^\beta$ are not necessarily edge-disjoint or vertex-disjoint. For any given non-dummy α-path or

β-path p, we use $\mathbb{P}^{\alpha}_{<p}$ to denote the set of all paths in $\mathbb{P}^{\alpha}$ whose lengths are smaller than the length of p. Similarly define $\mathbb{P}^{\beta}_{<p}$.

For any input X of Alice's, if node v has a stable failure in the simulation, we use the function $F_A(X, v)$ to denote v's failure time. Otherwise $F_A(X, v)$ is undefined. Similarly define the function $F_B(Y, v)$. For any path p and integer t, we use $p_A(X, t)$ to denote the existence of some node $v \in p$ satisfying $F_A(X, v) \leq t|p|$. Intuitively, this means that the path p will be cut in round $t|p|$ or earlier if Alice's input is X *and* if the execution continues up to round $t|p|$. We similarly define $p_B(Y, t)$. We will often drop the subscripts in $p_A(X, t)$ and $p_B(Y, t)$ since they are usually obvious. We say that $\mathbb{P}^{\alpha}_{<p}(X, t)$ holds if either $\mathbb{P}^{\alpha}_{<p}$ is empty or if $p'(X, t)$ holds for all $p' \in \mathbb{P}^{\alpha}_{<p}$. Similarly define $\mathbb{P}^{\alpha}_{<p}(Y, t)$, $\mathbb{P}^{\beta}_{<p}(X, t)$, and $\mathbb{P}^{\beta}_{<p}(Y, t)$. Also similarly define $\mathbb{P}^{\alpha}(X, t)$, $\mathbb{P}^{\alpha}(Y, t)$, $\mathbb{P}^{\beta}(X, t)$, and $\mathbb{P}^{\beta}(Y, t)$. For any given input pair $(X, Y) \in \mathcal{I}$, we say that a non-dummy α-path or β-path p is a *focal* path iff both of the following two properties hold:

- $\mathbb{P}^{\alpha}_{<p}(X, b)$, or $\mathbb{P}^{\alpha}_{<p}(Y, b)$, or both hold.
- $\mathbb{P}^{\beta}_{<p}(X, b)$, or $\mathbb{P}^{\beta}_{<p}(Y, b)$, or both hold.

Recall that b is the time complexity of the SUM protocol, in terms of flooding rounds. As we will prove later, a focal path p has the nice property that all α-paths and β-paths shorter than p will by cut by the end of the execution $\Phi(X, Y)$. This is often a necessary precondition for us to reason about various properties on p.

Finally, recall the definition of an flooding round from Chapter 2. We define $\lambda(X, Y)$ to be the number of rounds in an flooding round in the execution of $\Phi(X, Y)$. In other words, $\lambda(X, Y) = \max_{G' \in \mathcal{G}} \Lambda(G')$ where $\mathcal{G}$ is the set of topologies that have ever appeared in the execution of $\Phi(X, Y)$. Since an oblivious reduction needs to work for any arbitrary and black-box SUM oracle protocol whose time complexity is up to b flooding rounds for some given b, the oblivious reduction needs to work even under the worst case scenario where the execution of $\Phi(X, Y)$ takes as long as $b\lambda(X, Y)$ rounds.

Key challenge in the proof. Recall that Lemma 6.4.1 intends to claim that τ will be disconnected from the root in the execution of $\Phi(X, Y)$ for any $(X, Y) \in \mathcal{I}$. To prove

that τ will be disconnected from the root, we need to show that there does not exist any path in G (which can be arbitrary) for τ to reach the root when the execution ends. The key challenge here is that a failure in the reference setting for (X, Y) may or may not actually occur in the execution of $\Phi(X, Y)$ — if the execution terminates before the failure time of a node v, then node v does not actually fail in $\Phi(X, Y)$. This is further complicated by the fact that the total number of rounds in $\Phi(X, Y)$ (i.e., $b\lambda(X, Y)$) depends on the value of $\lambda(X, Y)$, which is itself affected by failures.

The above challenge implies that conceptually in our proof, we will need to start with an initial pessimistic guess (i.e., a loose lower bound) on $\lambda(X, Y)$ and on the total number of rounds in $\Phi(X, Y)$. Based on this pessimistic guess, we can show that certain failures must occur by the end of $\Phi(X, Y)$. Those failures in turn allow us to obtain a better guess on $\lambda(X, Y)$ (i.e., raising our lower bound on $\lambda(X, Y)$). This better guess then enables us to prove that some more failures will occur. Repeating this process (implicitly via an induction) will address the above challenge.

Some technical lemmas. In the next, we will prove a series of technical lemmas (Lemma 6.4.3 through 6.4.9), which will be need for the proof of Lemma 6.4.1 later.

Lemma 6.4.3. *Any path p from τ ($\tau \neq \alpha$ and $\tau \neq \beta$) to the root must contain an α-path or a β-path as a part.*

Proof. Trivially follows from the fact that α is the root. In fact, it is possible to prove the following stronger claim: p must either be an α-path itself or contains a β-path as a part. We chose to still state the lemma in its current form since we want the lemma to be symmetric for α and β. □

The following lemma says that if a path p is a focal path for an input pair $(X, Y) \in \mathcal{I}$, then $\lambda(X, Y)$ will be no smaller than the length of p. Intuitively, this holds because $\lambda(X, Y)$ is no smaller than the length of the shortest path from τ to α at the end of the execution $\Phi(X, Y)$. This shortest path must contain an α-path or a β-path as a part. But since p is a focal path, we will show that all α-paths and β-paths that are shorter than p must have been cut by the end of the execution. Hence this shortest path is no shorter than p.

Lemma 6.4.4. *Consider any focal path p for any input pair $(X, Y) \in \mathcal{I}$. We have $|p| \leq \lambda(X, Y)$.*

Proof. By the construction of $\mathcal{I}$, we know that for any input pair $(X, Y) \in \mathcal{I}$, τ has a value of 1 in the execution of $\Phi(X, Y)$. Lemma 6.3.1 tells us that τ will not be disconnected from the root in $\Phi(X, Y)$. Let p_1 denote the shortest path from τ to the root at the end of the execution $\Phi(X, Y)$. By definition of an flooding round, we know that the number of round in an flooding round is no smaller than the root's eccentricity in the graph, and thus we have $|p_1| \leq \lambda(X, Y)$. Next consider the set of all α-paths and β-paths. We claim that any α-path or β-path that is shorter than p will no longer exist (i.e., been cut) by the end of the execution $\Phi(X, Y)$. If this claim does hold, then notice that by Lemma 6.4.3, p_1 must contain an α-path or a β-path. This means that $|p| \leq |p_1| \leq \lambda(X, Y)$.

We prove the earlier claim via a contradiction, and assume that there exist some α-paths and/or β-paths that are shorter than p and they still exist at the end of the execution $\Phi(X, Y)$. Let p_2 be the shortest one of those paths (if there are multiple such p_2's, simply pick an arbitrary one). Note that p_2 must be a non-dummy path. Again since Lemma 6.4.3 tells us that p_1 must contain an α-path or a β-path, we must have $|p_2| \leq |p_1| \leq \lambda(X, Y)$. If $p_2 \in \mathbb{P}^{\alpha}$, then $p_2 \in \mathbb{P}^{\alpha}_{<p}$ since $|p_2| < |p|$. Since p is a focal path, we know that either $\mathbb{P}^{\alpha}_{<p}(X, b)$ or $\mathbb{P}^{\alpha}_{<p}(Y, b)$ hold, implying that either $p_2(X, b)$ or $p_2(Y, b)$ hold. Since $b|p_2| \leq b\lambda(X, Y)$, there will be a failure on p_2 by the end of the execution of $\Phi(X, Y)$. Contradiction. The case for $p_2 \in \mathbb{P}^{\beta}$ is similar. □

The following lemma says that for any input pair $(X, Y) \in \mathcal{I}$, $\mathbb{P}^{\alpha}(X, b)$ and $\mathbb{P}^{\beta}(Y, b)$ cannot both hold. Intuitively, this is because if they both held, then τ would be disconnected from the root (i.e., α) by the end of the execution $\Phi(X, Y)$.

Lemma 6.4.5. *For any input pair $(X, Y) \in \mathcal{I}$, it is impossible for $\mathbb{P}^{\alpha}(X, b)$ and $\mathbb{P}^{\beta}(Y, b)$ to both hold.*

Proof. By Lemma 6.3.1, we know that τ will not be disconnected from the root in the execution of $\Phi(X, Y)$. This means there exists some path p_1 from τ to the root at the end of the execution. Lemma 6.4.3 tells us that p_1 must contain an α-path or a β-path. This means that at the end of the execution, there is at least one α-path or β-path that has not been cut. Let p be the shortest α-path or β-path that has not been cut at the end of the execution (if there are multiple such p's, simply pick an arbitrary one). Clearly p must be a non-dummy path.

First consider the case where p is a non-dummy α-path. By how we pick p, we know that p is a focal path for (X, Y). Lemma 6.4.4 tells us that $|p| \leq \lambda(X, Y)$. Now since p has not been cut at the end of the execution in round $b\lambda(X, Y)$, we know that $p(X, b)$ must not hold. This means that $\mathbb{P}^{\alpha}(X, b)$ does not hold.[3] If p is a non-dummy β-path, then one can similarly show that $\mathbb{P}^{\beta}(Y, b)$ does not hold. □

The next lemma considers any given focal path p with respect to an input pair $(X, Y) \in \mathcal{I}$. The lemma intuitively says that if some node in p is already spoiled for Alice's input X in a certain round, then this spoiled node will cause all other nodes in p to become spoiled within the next $|p|$ rounds, unless a stable failure is simulated on the path to "block" such spreading of spoiled nodes.

Lemma 6.4.6. *Consider any focal path p for any input pair $(X, Y) \in \mathcal{I}$. In the execution of $\Phi(X, Y)$, for all $t \leq b - 1$:*

- *If some node in p is spoiled for Alice's input X in round $t|p|$ and if $p(X, t + 1)$ does not hold, then all nodes in p are spoiled for Alice's input X in round $(t + 1)|p|$.*
- *If some node in p is spoiled for Bob's input Y in round $t|p|$ and if $p(Y, t+1)$ does not hold, then all nodes in p are spoiled for Bob's input Y in round $(t + 1)|p|$.*

Proof. First, Lemma 6.4.4 tells us that $|p| \leq \lambda(X, Y)$, and thus $(t + 1)|p| \leq b\lambda(X, Y)$. The remainder of the proof follows directly from the definition of spoil paths. In particular, by definition of spoil paths, only stable failures can block spoil paths. □

The next lemma still considers any given focal path p with respect to an input pair $(X, Y) \in \mathcal{I}$. The lemma intuitively says that if some node on p has a stable failure with respect to Y and is not simultaneously a stable failure with respect to X, then that node becomes an epicenter for Alice. Unless we simulate a stable failure (with respect to X) on p to "block" the spreading of spoiled nodes caused by this epicenter, all nodes on p will be spoiled for Alice's input X within the next $|p|$ rounds.

Lemma 6.4.7. *Consider any focal path p for any input pair $(X, Y) \in \mathcal{I}$. In the execution of $\Phi(X, Y)$, for all $t \leq b - 1$:*

[3] One can also simultaneously show that $\mathbb{P}^{\alpha}(Y, b)$ does not hold, though we do not need that claim.

- *If $p(Y,t)$ holds and if $p(X,t+1)$ does not hold, then all nodes in p are spoiled for Alice's input X in round $(t+1)|p|$.*
- *If $p(X,t)$ holds and if $p(Y,t+1)$ does not hold, then all nodes in p are spoiled for Bob's input Y in round $(t+1)|p|$.*

Proof. First, Lemma 6.4.4 tells us that $|p| \leq \lambda(X,Y)$, and thus $(t+1)|p| \leq b\lambda(X,Y)$. Without loss of generality, we only prove the first part of the lemma. By definition of $p(Y,t)$, we know that there exists some node $v \in p$ such that $F_B(Y,v) \leq t|p| < (t+1)|p| \leq b\lambda(X,Y)$. Since $p(X,t+1)$ does not hold, the failure of v in round $F_B(Y,v)$ must not be a stable failure for Alice's input X. But since v does fail in the reference setting for (X,Y) in round $F_B(Y,v)$, it means that v is an epicenter for Alice's input X. (Note that v may still be either a value epicenter or a failure epicenter.) The occurrence time of this epicenter is round $F_B(Y,v)$ or earlier. This means that v must be spoiled in round $F_B(Y,v) \leq t|p|$. Apply Lemma 6.4.6 then finishes the proof. □

The next lemma still considers any given focal path p with respect to an input pair $(X,Y) \in \mathcal{I}$. The lemma intuitively says that if τ is spoiled for Alice's input X, in order to prevent α from being spoiled within th next $|p|$ rounds, we must simulate a stable failure with respect to X somewhere on p.

Lemma 6.4.8. *Consider any focal path p for any input pair $(X,Y) \in \mathcal{I}$. For all $t \leq b-1$:*

- *If $p \in \mathbb{P}^\alpha$ and if τ is spoiled for Alice's input X in round $t|p|$, then $p(X,t+1)$ must hold.*
- *If $p \in \mathbb{P}^\beta$ and if τ is spoiled for Bob's input Y in round $t|p|$, then $p(Y,t+1)$ must hold.*

Proof. First, Lemma 6.4.4 tells us that $|p| \leq \lambda(X,Y)$, and thus $(t+1)|p| \leq b\lambda(X,Y)$. Without loss of generality, we only prove the first part, via a contradiction. By Lemma 6.4.6, if $p(X,t+1)$ does not hold, then in the execution of $\Phi(X,Y)$, all nodes in p are spoiled for Alice's input X in round $(t+1)|p|$. Since $(t+1)|p| \leq b\lambda(X,Y)$, this means that α (which is in p) is spoiled by the end of the execution, which contradicts with Corollary 6.4.1. □

The following will use the above lemmas to prove the final technical lemma in this section. Even though the proof only uses elementary induction, it is rather complex because while we are doing an induction on the input pairs in the problematic input set $\mathcal{I}$, we need to simultaneously reason about the multiple paths in G and these two issues are entangled together. Later we will only need to use the second claim in the following lemma — the first claim in the lemma is proved so that we can carry both claims in the induction, which is critical for the proof to work.

Lemma 6.4.9. *Suppose $b \geq 27$. Let $X^{(0)}$, $Y^{(1)}$, $X^{(1)}$, $Y^{(2)}$, ..., $X^{(k)}$, $Y^{(k+1)}$ (where $k+1 \leq \lfloor \sqrt{b/3} \rfloor$) be those inputs in the proof of Lemma 6.3.2 that correspond to the problematic input set $\mathcal{I}$. For any path $p \in \mathbb{P}^{\alpha}$, any integer $i \in [1, k+1]$, and any integer $t_i \in [0, b - 2i^2 - 2i]$, we have:*

- *If $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, t_i + 4i)$ and $\mathbb{P}^{\beta}_{<p}(Y^{(i)}, t_i)$ holds, then $p(X^{(i-1)}, t_i + 4i)$ holds.*
- *In particular, if $\mathbb{P}^{\alpha}_{<p} = \mathbb{P}^{\beta}_{<p} = \emptyset$, then $p(X^{(i-1)}, t_i + 4i)$ holds.*

Proof. First, note that $i \leq k+1 \leq \lfloor \sqrt{b/3} \rfloor$ and $b \geq 27$ imply $b - 2i^2 - 2i \geq 0$. This means that the range for t_i is never empty. We only prove the first part of the lemma, since the second part is the special case of the first part. We prove the first part via an induction on i. For $i = 1$, since $t_1 < t_1 + 4 \leq b$, we have $\mathbb{P}^{\alpha}_{<p}(X^{(0)}, b)$ and $\mathbb{P}^{\beta}_{<p}(Y^{(1)}, b)$. This means that p is a focal path for the input pair $(X^{(0)}, Y^{(1)})$. In the execution of $\Phi(X^{(0)}, Y^{(1)})$, τ has a value of 1 and is spoiled for Alice's input $X^{(0)}$ in round $1 \leq |p|$. Apply Lemma 6.4.8 and we have $p(X^{(0)}, 2)$, which implies $p(X^{(0)}, t_1 + 4)$ for all $t_1 \in [0, b-4]$.

Now consider any $i \geq 2$, while assuming that the lemma holds for $i - 1$. We are given the condition $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, t_i + 4i)$ and $\mathbb{P}^{\beta}_{<p}(Y^{(i)}, t_i)$. We will prove $p(X^{(i-1)}, t_i + 4i)$ via a contradiction and assume that it does not hold. The final contradiction will be obtained by sequentially proving the following claims:

- **Claim 1:** $\mathbb{P}^{\beta}_{<p}(X^{(i-1)}, t_i + 1)$ holds, which will be proved via the execution of $\Phi(X^{(i-1)}, Y^{(i)})$.
- **Claim 2:** $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$ holds, which will be proved via the execution of $\Phi(X^{(i-1)}, Y^{(i-1)})$.

- **Claim 3:** $\mathbb{P}^{\alpha}_{<p}(X^{(i-2)}, t_i + 4i - 2)$ holds, which will be proved via the inductive hypothesis and implicitly via the execution of $\Phi(X^{(i-2)}, Y^{(i-1)})$.
- **Claim 4:** $p(X^{(i-2)}, t_i + 4i - 2)$ holds, which will be proved via the inductive hypothesis and implicitly via the execution of $\Phi(X^{(i-2)}, Y^{(i-1)})$.
- **Claim 5:** $p(Y^{(i-1)}, t_i + 4i - 1)$ holds, which will be proved via the execution $\Phi(X^{(i-2)}, Y^{(i-1)})$.
- **Claim 6:** $p(X^{(i-1)}, t_i + 4i)$ holds, which will be proved via the execution of $\Phi(X^{(i-1)}, Y^{(i-1)})$.

Figure 6.2 illustrates these 6 claims in an example topology.

Proving Claim 1. We prove $\mathbb{P}^{\beta}_{<p}(X^{(i-1)}, t_i + 1)$ via a contradiction and let p_1 be any path in $\mathbb{P}^{\beta}_{<p}$ where $p_1(X^{(i-1)}, t_i + 1)$ does not hold. We first show that p and p_1 are both focal paths for the input pair $(X^{(i-1)}, Y^{(i)})$. For p, we have $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, t_i + 4i)$ and $\mathbb{P}^{\beta}_{<p}(Y^{(i)}, t_i)$. Since $t_i < t_i + 4i < b$, we know that p is a focal path for $(X^{(i-1)}, Y^{(i)})$. For p_1, since $\mathbb{P}^{\alpha}_{<p_1} \subset \mathbb{P}^{\alpha}_{<p}$ and $\mathbb{P}^{\beta}_{<p_1} \subset \mathbb{P}^{\beta}_{<p}$, by similar argument we know that p_1 is a focal path for $(X^{(i-1)}, Y^{(i)})$ as well.

Next by the original condition, we have $p_1(Y^{(i)}, t_i)$. Invoke Lemma 6.4.7 for p_1 and we know that all nodes on p_1 (including τ) are spoiled for Alice's input $X^{(i-1)}$ in the execution of $\Phi(X^{(i-1)}, Y^{(i)})$ in round $(t_i + 1)|p_1| < (t_i + 1)|p|$. We next invoke Lemma 6.4.8 for p and we know that $p(X^{(i-1)}, t_i+2)$ must hold, which implies $p(X^{(i-1)}, t_i+4i)$. Contradiction.

Proving Claim 2. Consider any path $p_1 \in \mathbb{P}^{\beta}_{<p}$. We first show that p_1 is a focal path for the input pair $(X^{(i-1)}, Y^{(i-1)})$. From the original condition of $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, t_i + 4i)$ and since $\mathbb{P}^{\alpha}_{<p_1} \subset \mathbb{P}^{\alpha}_{<p}$, we have $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-1)}, t_i + 4i)$ which implies $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-1)}, b)$. Next Claim 1 tells us that $\mathbb{P}^{\beta}_{<p}(X^{(i-1)}, t_i + 1)$. By a similar argument, we have $\mathbb{P}^{\beta}_{<p_1}(X^{(i-1)}, b)$. Thus p_1 is a focal path for $(X^{(i-1)}, Y^{(i-1)})$.

From Claim 1, we also have $p_1(X^{(i-1)}, t_i + 1)$. Now invoke Lemma 6.4.7 for p_1. We will then have $p_1(Y^{(i-1)}, t_i + 2)$, since otherwise all nodes in p_1 (including β) are spoiled for Bob's input $Y^{(i-1)}$ in the execution of $\Phi(X^{(i-1)}, Y^{(i-1)})$ in round $(t_i + 2)|p_1|$.

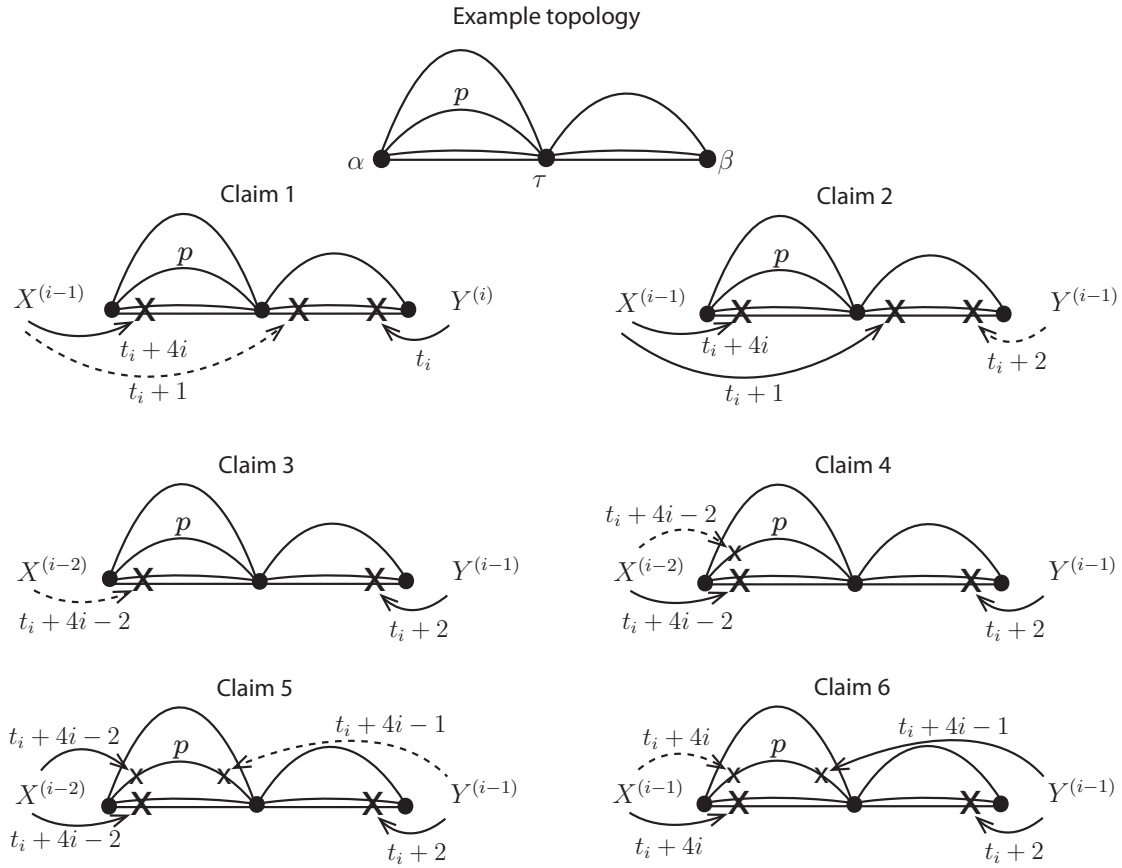

Figure 6.2: Illustration of the 6 claims proved in Lemma 6.4.9 in an example topology. For the given τ, this example topology has 4 α-paths and 3 β-paths. Nodes other than α, β, and τ are not shown in the figure. The α-path marked by p is the path p in the lemma. For clarity, we omit the labels for α, β, and τ when illustrating the claims. For each of the claims, the figure indicates on the left the input (e.g., $X^{(i-1)}$) to Alice, and on the right the input to Bob. Solid arrows indicate those (stable) failures that we already know, given the corresponding input to Alice or Bob. Dashed arrows indicate those (stable) failures whose existence is proved in the corresponding claim.

Proving Claim 3. Prove by contradiction and assume that $\mathbb{P}^{\alpha}_{<p}(X^{(i-2)}, (t_i+2)+4(i-1))$ does not hold. Let p_1 be the shortest path (if there are multiple such p_1's, simply pick an arbitrary one) in $\mathbb{P}^{\alpha}_{<p}$ such that the $p_1(X^{(i-2)}, (t_i + 2) + 4(i - 1))$ does not hold. We next want to invoke the inductive hypothesis for $i - 1$ on $p_1 \in \mathbb{P}^{\alpha}$ with $t_{i-1} = t_i + 2$. Such invocation is possible since:

- $t_{i-1} = t_i + 2 \leq b - 2i^2 - 2i + 2 < b - 2i^2 - 2i + 4i = b - 2(i - 1)^2 - 2(i - 1)$.
- By definition of p_1, we have that $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-2)}, (t_i + 2) + 4(i - 1))$ holds.
- We know from Claim 2 that $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$ holds, implying that $\mathbb{P}^{\beta}_{<p_1}(Y^{(i-1)}, t_i + 2)$ holds.

This invocation tells us that $p_1(X^{(i-2)}, (t_i + 2) + 4(i - 1))$ holds, leading to a contradiction.

Proving Claim 4. We want to invoke the inductive hypothesis for $i - 1$ on p with $t_{i-1} = t_i + 2$. Such invocation is possible since:

- $t_{i-1} = t_i + 2 \leq b - 2i^2 - 2i + 2 < b - 2i^2 - 2i + 4i = b - 2(i - 1)^2 - 2(i - 1)$.
- Claim 3 gives us $\mathbb{P}^{\alpha}_{<p}(X^{(i-2)}, (t_i + 2) + 4(i - 1))$.
- Claim 2 gives us $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$.

This invocation tells us that $p(X^{(i-2)}, (t_i + 2) + 4(i - 1))$ holds.

Proving Claim 5. We first show that p is a focal path for $(X^{(i-2)}, Y^{(i-1)})$. We already have $\mathbb{P}^{\alpha}_{<p}(X^{(i-2)}, t_i + 4i - 2)$ from Claim 3 and $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$ from Claim 2. Since $t_i + 2 \leq t_i + 4i - 2 \leq b - 2i^2 - 2i + 4i - 2 < b$, we now know that p is a focal path for $(X^{(i-2)}, Y^{(i-1)})$. We next prove $p(Y^{(i-1)}, t_i + 4i - 1)$ via a contradiction.

We already have $p(X^{(i-2)}, t_i + 4i - 2)$ from Claim 4. Since $p(Y^{(i-1)}, t_i + 4i - 1)$ does not hold, we can invoke Lemma 6.4.7 for p. That lemma tells us that in the execution of $\Phi(X^{(i-2)}, Y^{(i-1)})$, all nodes in p (including τ) become spoiled for Bob's input $Y^{(i-1)}$ in round $(t_i + 4i - 1)|p|$. This is a critical property which we will use later.

Next consider the two properties $\mathbb{P}^{\alpha}(X^{(i-2)}, t_i + 8i - 4)$ and $\mathbb{P}^{\beta}(Y^{(i-1)}, t_i + 4i)$. By Lemma 6.4.5, it is impossible for both of them to hold, since otherwise they would imply that both $\mathbb{P}^{\alpha}(X^{(i-2)}, b)$ and $\mathbb{P}^{\beta}(Y^{(i-1)}, b)$ hold. Let p_1 be the shortest path (if there are multiple such p_1's, simply pick an arbitrary one) in $\mathbb{P}^{\alpha} \cup \mathbb{P}^{\beta}$ where $p_1(X^{(i-2)}, t_i + 8i - 4)$ (if $p_1 \in \mathbb{P}^{\alpha}$) or $p_1(Y^{(i-1)}, t_i + 4i)$ (if $p_1 \in \mathbb{P}^{\beta}$) does not hold.

We consider two cases. If $p_1 \in \mathbb{P}^{\beta}$, we will first show that p_1 is a focal path. By definition of p_1, we have $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-2)}, t_i + 8i - 4)$ and $\mathbb{P}^{\beta}_{<p_1}(Y^{(i-1)}, t_i + 4i)$ holds. Since $t_i + 4i \leq b - 2i^2 - 2i + 4i < b$, $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-2)}, b)$ and $\mathbb{P}^{\beta}_{<p_1}(Y^{(i-1)}, b)$ holds. This means that p_1 is a focal path for the input pair $(X^{(i-2)}, Y^{(i-1)})$. We next want to show that $|p| \leq |p_1|$. By how we chose p_1, we know that $p_1(Y^{(i-1)}, t_i + 4i)$ does not hold. On the other hand, Claim 2 tells us that $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$ holds, implying that $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 4i)$ holds (since $i \geq 2$). Thus we must have $p_1 \notin \mathbb{P}^{\beta}_{<p}$ and $|p_1| \geq |p|$. Finally, as shown earlier, in the execution of $\Phi(X^{(i-2)}, Y^{(i-1)})$, the node τ must be spoiled for Bob's input $Y^{(i-1)}$ in round $(t_i + 4i - 1)|p| \leq (t_i + 4i - 1)|p_1|$. Now we can invoke Lemma 6.4.8, which shows that $p_1(Y^{(i-1)}, t_i + 4i)$ holds and thus leads to a contradiction.

For the second case where $p_1 \in \mathbb{P}^{\alpha}$, we want to invoke the inductive hypothesis for $i - 1$ on p_1 with $t_{i-1} = t_i + 4i$. Such invocation is possible since:

- $t_{i-1} = t_i + 4i \leq b - 2i^2 - 2i + 4i = b - 2(i-1)^2 - 2(i-1)$.
- By definition of p_1, we have that $\mathbb{P}^{\alpha}_{<p_1}(X^{(i-2)}, (t_i + 4i) + 4(i-1))$ and $\mathbb{P}^{\beta}_{<p_1}(Y^{(i-1)}, t_i + 4i)$ holds.

The invocation gives us $p_1(X^{(i-2)}, (t_i + 4i) + 4(i-1))$, leading to a contradiction.

Proving Claim 6. We first show that p is a focal path for $(X^{(i-1)}, Y^{(i-1)})$. From the original condition, we have $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, t_i + 4i)$, which implies $\mathbb{P}^{\alpha}_{<p}(X^{(i-1)}, b)$. By Claim 2, we have $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, t_i + 2)$, which implies $\mathbb{P}^{\beta}_{<p}(Y^{(i-1)}, b)$. Thus p is a focal path for $(X^{(i-1)}, Y^{(i-1)})$.

Claim 5 gives us $p(Y^{(i-1)}, t_i + 4i - 1)$. Now invoke Lemma 6.4.7 for p. That lemma tells us that $p(X^{(i-1)}, t_i + 4i)$ must hold, since otherwise all nodes in p (including α) will be spoiled for Alice's input $X^{(i-1)}$ in round $(t_i + 4i)|p|$. □

6.4.3 Proof for Lemma 6.4.1

Using the technical lemmas proved in the previous section, we can now finally prove Lemma 6.4.1:

for Lemma 6.4.1. By Lemma 6.4.3, a path from τ to the root must contain either an α-path or a β-path. Thus to prove the lemma, it suffices to prove that all α-paths and β-paths are dummy. Prove by contradiction and assume that some α-paths and/or β-paths are non-dummy. Let p be the shortest path among all such paths (if there are multiple such p's, simply pick an arbitrary one). This means that there exists some $(X, Y) \in \mathcal{I}$ such that p is not cut by the end of the execution $\Phi(X, Y)$. Also by how we chose p, we trivially have $\mathbb{P}^{\alpha}_{<p} = \mathbb{P}^{\beta}_{<p} = \emptyset$.

Next first consider the case where p is a non-dummy α-path. For all i where $1 \le i \le k + 1 \le \lfloor \sqrt{b/3} \rfloor$, invoke the second claim in Lemma 6.4.9 with $t_i = 0$ and we have $p(X^{(i-1)}, 4i)$. Since $4i \le 4\lfloor \sqrt{b/3} \rfloor < b$ when $b \ge 27$, this in turn implies $p(X^{(i-1)}, b)$ for $1 \le i \le k + 1$. Next since $\mathbb{P}^{\alpha}_{<p} = \mathbb{P}^{\beta}_{<p} = \emptyset$, we trivially know that p is a focal path for (X, Y). Invoke Lemma 6.4.4 and we have $|p| \le \lambda(X, Y)$. Together with $p(X, b)$, we know that p will be cut by the end of the execution of $\Phi(X, Y)$. Contradiction.

For the second case where p is a β-path, the proof is entirely symmetric. In particular, the only difference between α and β is that α is the root while β is not. However, we only used the fact that α is the root in the proof of Lemma 6.4.3. Lemma 6.4.3 itself is already symmetric for α and β. In other words, if we view the proof for Lemma 6.4.3 as a black-box, then α and β are entirely symmetric throughout Sections 6.4.1 and 6.4.2. □

Chapter 7

Lower Bounds on FT Communication Complexity of SUM for All b

Our previous FT lower bounds from Chapter 4 become trivial when $b \geq \frac{f}{\log N}$ or $\sqrt{\frac{1}{\epsilon \log n}}$. Chapter 6 have suggested that such limitation might be inherent in the approach used in Chapter 4. This section uses a different approach to obtain logarithmic FT lower bounds for such b, which is more than exponentially far away from the corresponding $O(1)$ NFT upper bounds for such b. Specifically, this section aims to eventually prove the following theorem:

Theorem 1.4.2 (Restated)**.** *For any $b \geq 1$, we have:*

$$\begin{aligned} \mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, 2N, b) &= \Omega(\log N) \\ \mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) &= \Omega\left(\log \frac{1}{\epsilon}\right), \quad \textit{for } \epsilon = \Omega\left(\frac{1}{N}\right) \end{aligned}$$

Note that i) under sufficiently large b, a message of any given size can be encoded using a single bit. Hence $\Omega(\log N)$ and $\Omega(\log \frac{1}{\epsilon})$ actually lower bound the number of messages, and the theorem can only be proved by reasoning about the number of messages; ii) we focus on $\epsilon = \Omega(\frac{1}{N})$ since if the summation is $O(N)$, there isn't much interesting to study the case where $\epsilon = o(\frac{1}{N})$.

In the following, Section 7.1 first gives some intuitions and reveals the key challenge in proving the theorem. Next Section 7.2 describes the topology and the failure ad-

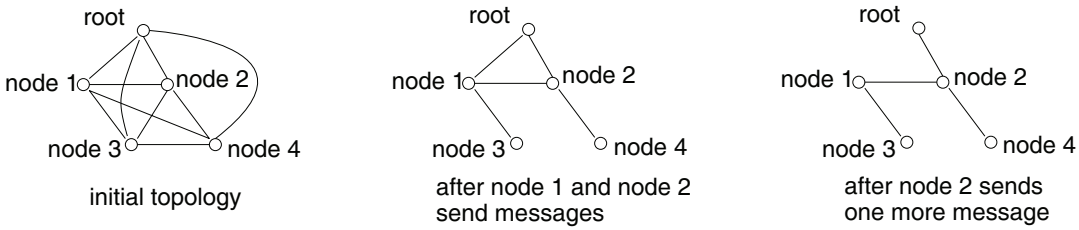

Figure 7.1: Example FT lower bound topology for $n = 4$ and unrestricted b.

versary used for proving the theorem. Section 7.3 defines a simple single-player *probing game*, and then proves a connection (or a reduction in the general sense) between the probing game and the SUM problem. This is the key step since it enables us to focus on lower bounding the "performance" of the probing game, which is much easier to study than the SUM problem. Section 7.4 then proves a lower bound on the "performance" of the probing game. This lower bound, together with the connection established earlier, enables us to prove Theorem 1.4.2 finally in Section 7.5.

7.1 Obtaining Some Intuitions under the Gossip Assumption

We first provide some intuitions for Theorem 1.4.2 under a strong *gossip assumption*. Doing so also helps to reveal the key technical challenges to be addressed by our proof later.

Under the *gossip assumption*, the root computes the sum by explicitly collecting from each node a gossip containing its value. We will intuitively show that to do so, some node will need to send $\Omega(\log N)$ messages, and hence $\Omega(\log N)$ bits even if the gossips can be fully aggregated/compressed. Here the lower bound topology will be an N-node clique with one of nodes being the root (Figure 7.1). Imagine for now that the adversary can fail edges in this topology, and further there is never more than one node sending messages in a round. We also assume that the number of edge failures can be larger than $\Theta(N)$. These assumptions can be easily removed later once we insert some dummy nodes into each edge. Our adaptive adversary

waits until exactly $\frac{N-1}{2}$ non-root nodes have sent a message (e.g., nodes 1 and 2 in Figure 7.1). Call these $\frac{N-1}{2}$ nodes as *marked* nodes. The adversary then fails enough edges so that each unmarked non-root node (e.g., node 3) is paired up with a marked node (e.g., node 1) and the marked node is the only gateway for the unmarked node to reach the root. Now each marked nodes has already sent a message, and yet it has one new gossip (from the corresponding unmarked node) to forward to the root. Next apply this procedure recursively on these $\frac{N-1}{2}$ marked nodes, and inject a second batch of edge failures when exactly $\frac{N-1}{4}$ of them (e.g., node 2) have sent a second message. Continuing this argument can easily show that for all the gossips to reach the root, some node needs to send at least $\log(N-1)+1$ messages.

Formalizing the above arguments would be sufficient to prove Theorem 1.4.2, if the gossip assumption were valid. Unfortunately, the gossip assumption is actually rather strong and nodes can communicate "via silence". For example, a protocol may be such that if a node's value is 0, then the root does not need to collect a gossip from that node and simply uses 0 as the default value. It is also possible that node i sends a message to node j iff node i's value is 1, and then node j conceptually relays i's value to the root, by sending a message to the root iff this value is 0. Here the root *never* collects a gossip from node i. Properly capturing all such possibilities will be the key challenge in our proof.

7.2 Topology and Adversary for Proving Theorem 1.4.2

Having explained the basic intuition and challenges, we now construct the topology and failure adversary for later proving Theorem 1.4.2. We assume that each node can take an integer value in the domain of $[0, 3n-1]$ — this assumption will be removed later in Section 7.5. We construct the lower bound topology starting from a clique with $n+1$ nodes, with n being a power of 2. One of these nodes will be the root, while all other nodes are called *worker* nodes. We next insert a degree-two *dummy node* in the middle of each edge in the previous clique, so that failing the dummy node essentially fails the corresponding edge. The topology thus has total $\frac{(n+1)(n+2)}{2}$ nodes. Each worker node has an integer value in $[0, 3n-1]$. All other nodes have

value of 0. We use a vector $W = (w_1, w_2, ..., w_n)$ to denote the *input* (to the system), where $w_i \in [0, 3n-1]$ is the input value of worker node i.

Our deterministic and adaptive failure adversary here is the same as the simple failure adversary in the previous section, except that the adversary here fails the dummy nodes instead of failing the edges. (Since all dummy nodes are of degree 2, the number of edges incidental to failed nodes is at most $2N$.) Specifically, our adversary here conceptually partitions the worker nodes into groups, where each group has some *group members* and one group member is the *group leader*. Group membership and leadership change whenever the adversary inject failures. Initially each worker is in its own group, with itself being the sole member (and thus the leader). The adversary keeps track of the set L of current leaders. A leader in L is *marked* once it sends a message. Once the number of marked leaders reaches $\frac{|L|}{2}$, the adversary pairs up each unmarked leader node j with a distinct marked leader node i. In cases where multiple leaders send messages in the same round, the adversary will mark them sequentially (by their ids), until the number of marked leaders becomes exactly $\frac{|L|}{2}$. Next for each such node j, consider each of its neighboring dummy nodes. The dummy node may connect j with i) the root, ii) some leader node other than node i, iii) leader node i, or iv) some non-leader node. If the dummy node connects j with the root or with some leader node other than node i, then at the beginning of the next round, the adversary fails that dummy node. After all these failures are injected, node i's group and node j's group are conceptually merged into one new group, with node i being the new group leader. Finally, the adversary updates L to be the set of those $\frac{|L|}{2}$ leaders of the new groups, clears all the marks on those leaders, and repeats the above process until $|L|$ reaches 1 or the protocol terminates.

7.3 The Probing Game and Its Connection to SUM

We next define a simple *probing game*. We will draw a connection (or a reduction in the general sense) between this game and *deterministic* SUM protocols, when we run such SUM protocols under the topology and failure adversary as constructed in the previous section. Later in Section 7.5, we will use some well-known technique to establish a connection between randomized and deterministic SUM protocols.

The *probing game* is played by a single player, against an input $W = (w_1, w_2, ..., w_n)$ where w_i is an integer in $[0, 3n - 1]$. W is initially not known to the player, but the player knows n. The player proceeds in rounds. In each round, the player may choose to sequentially do zero, one, or multiple *probes*, where each probe is in the form of a tuple (i, j). The outcome of the probe (i, j) is a *hit* if $w_i = j$. Otherwise it is a *miss*. For each probe, the player may *adaptively* choose what probe to do, based on the probing outcomes in previous rounds and so far in the current round. The goal of the player is to determine $\sum_{i=1}^{n} w_i$ based on the outcomes of the probes, while minimizing the total number of hits. Note that the player is not concerned with the total number of probes. For convenience later, we require that the player never does the same probe multiple times. In addition, if there has been a hit (i, j), then the player does not further probe (i, j') for any j' since the player has already learned w_i precisely. Clearly, if the player is required to output correctly for every input pattern, there is a trivial lower bound $\Omega(n)$ on the number of hits. Later in Section 7.4, we will show this lower bound continue to hold even if the player only need to output correctly in $2/3$ fraction of all possible inputs.

Consider any *deterministic* SUM protocol running under the topology and adversary as constructed in the previous section. The protocol can obviously play various tricks to minimize communication (e.g., by communicating "via silence"). But Theorem 7.3.1 below reveals that what the protocol fundamentally can do is no different from a virtual player playing the probing game and upon a hit, having some leader node in the topology send a message. In turn, the total number of messages sent by the leaders will be no smaller than the number of hits in the probing game.

Theorem 7.3.1. *Given any deterministic* SUM *protocol $\mathcal{P}$, there always exists a deterministic (adaptive) probing strategy $\mathcal{S}$ for the player in the probing game that satisfies the following property. For any input W, the player using $\mathcal{S}$ in the probing game against W always generates the same result as the result generated by the* SUM *protocol $\mathcal{P}$ running against W under the topology and adversary in Section 7.2. Furthermore, if the total number of hits in the probing game reaches n when using $\mathcal{S}$ against W, then the maximum number of bits sent by a node (across all nodes), when running $\mathcal{P}$ against W under the topology and adversary in Section 7.2, is at least* $\log n + 1$.

Proof. We will construct $\mathcal{S}$ based on the given (black-box) deterministic protocol

$\mathcal{P}$. The constructed strategy $\mathcal{S}$ will determine the sequence of probes that the player should do in each round r, so that the probing outcomes will enable the player to simulate the execution of $\mathcal{P}$ in round r. During the course of the probing game, we say that a worker node i has been *hit* if there has been some probe (i, j) that is a hit. In any given round r of $\mathcal{P}$'s execution, we say that a group leader node i sends an *influential message* in round r if node i sends (i.e., locally broadcasts) a message in round r and the adversary does not fail the dummy node connecting node i with the root at the beginning of round $r+1$. Note that since node i is a group leader in round r, it is guaranteed (by design of our adversary) that the dummy node connecting i with the root has not failed in round r or earlier. If the adversary indeed fails that dummy node at the beginning of round $r+1$, then node i will no longer be a group leader in round $r+1$. Furthermore, node i (and node i's group) will be merged with another group, with another node being the new (merged) group's leader.

We will prove that when the player uses our constructed probing strategy $\mathcal{S}$, all the following properties hold for all round r where $0 \leq r \leq R$. Here R is the total number of rounds in $\mathcal{P}$'s execution over W, which must be finite. Recall that round 1 is the first round where there can possibly be a message sent in $\mathcal{P}$'s execution.

Property 1 In round r of the probing game, if the player does a probe (i, j) and if this probe is a hit, then in round r of $\mathcal{P}$'s execution, nodes i's group leader must send an influential message.

Property 2 In $\mathcal{P}$'s execution, if a group leader sends an influential message in round r, then all nodes in that group have been hit by the player in the probing game by the end of round r.

Property 3 In $\mathcal{P}$'s execution, immediately after the adversary injects potential failures at the beginning of round $r+1$, in each group there is at most one worker node that has not been hit by the player in the probing game.[1]

Property 4 In round r, the player in the probing game can generate all influential messages sent by all group leaders in round r of $\mathcal{P}$'s execution.

[1] First, we explicitly mention "after failure injection" since group membership is affected by failures. Second, we consider the beginning of round $r+1$ instead of the beginning of round r to facilitate our later proof by induction.

Lemma 7.3.1 below proves that the above 4 properties indeed hold under a properly constructed probing strategy $\mathcal{S}$. Now by Property 4, since the influential messages from group leaders are the only messages that can affect the root via the dummy nodes, the player will be able to simulate those dummy nodes and all incoming messages to the root throughout the execution. Then the root will be able to produce a final result in round R, and the player simply uses this result as the result for the probing game. Next if the total number of hits in the probing game reaches n, then all nodes must have been hit since each node can only contribute one hit. By Property 3, we know that in any round, each group can have at most one worker node that has not been hit. Initially there are n groups, and thus there must exist at least $\frac{n}{2}$ groups where each group contributes a hit. By Property 1, the $\frac{n}{2}$ leaders of these $\frac{n}{2}$ groups will each send an influential message. Once all these influential messages are sent, our adversary will introduce failures so that there will be $\frac{n}{2}$ new groups, with these $\frac{n}{2}$ nodes as new leaders. Since we still need to have $\frac{n}{2}$ more hits and since each group can contribute at most one hit, among those $\frac{n}{2}$ groups, there must exist $\frac{n}{4}$ groups whose leaders will each send a second influential message. Continuing such argument will show that in order for the total number of hits in the probing game to reach n, some node in the SUM protocol $\mathcal{P}$ will have to send at least a influential messages, where a is the total number of terms in the summation of $n = \frac{n}{2} + \frac{n}{4} + ... + 2 + 1 + 1$. Observing that $a = \log_2 n + 1$ and that a messages translate to at least a bits completes the proof. □

Lemma 7.3.1. *Under the conditions of Theorem 7.3.1, there exists a probing strategy $\mathcal{S}$ such that the 4 properties described in the proof of Theorem 7.3.1 hold for all round r where $0 \leq r \leq R$. Here R is the total number of rounds in $\mathcal{P}$'s execution over W.*

Proof. We prove via an induction on r. For $r = 0$, we construct $\mathcal{S}$ such that no probes are done in round 0. The 4 properties trivially hold for round 0. Now consider any round $r > 0$, while assuming that they hold for all rounds before r.

We first construct the set of probes that the player should do in round r. Consider any group g in round r of $\mathcal{P}$'s execution, after the adversary injects potential failures at the beginning of round r. (Note that failures affect group membership, and thus we explicitly state that g is defined after the failures have been injected in round r.) By inductive hypothesis on Property 3, there is at most one worker node $miss(g)$ in

g that has not been hit. Thus the player knows the value of all other nodes in g. If $miss(g)$ exists, then the player will exhaustively enumerate all j's such that there has not been a probe $(miss(g), j)$. Intuitively, j has not been ruled out as a possible value for $miss(g)$. For each such j, the player *tries* simulating $\mathcal{P}$'s execution on all nodes in g, from round 0 to round r (inclusive). Doing so will enable the player to determine the message (if any) sent in round r by g's group leader. Such simulation is possible since the player has the values of all the nodes in that group, and also because by inductive hypothesis on Property 4, the player can generate all influential messages sent by other group leaders up to round $r-1$. In particular for a node i in g, all incoming messages from nodes outside of g must be sent from those dummy nodes connecting node i with those group leaders in round 1 through round $r-1$. The reason is that in any round from 1 through $r-1$, non-leader group members can only have neighboring dummy nodes connecting them to other nodes in their own groups and not to node i. Furthermore, non-influential messages from a group leader can never affect either the root or nodes in other groups (via the corresponding dummy nodes), since those dummy nodes will be failed right after they receive those non-influential messages. Finally, we do not yet know what influential messages other group leaders will send in round r, but those will not affect the potential message sent in round r by g's group leader.

We say that a $(miss(g), j)$ combination is a *candidate probe* if $miss(g)$ exists and by the above process, the player has determined that there will be a message sent in round r by g's group leader *if* j is the value of node $miss(g)$. (We do not yet know whether this message will be influential.) Next the player orders all candidate probes, using g as the primary key and j as the secondary key. In round r, the player sequentially issues the probes in this ordered list, subject to the following two constraints. First, if a certain $(miss(g), j)$ probe is a hit, then the player will skip all following probes in the form of $(miss(g), j')$ for all j'. Second, if the number of hits so far is such that the adversary is ready to inject the next batch of failures, the player skips all the remaining probes in the ordered list.

We next prove that the probing strategy constructed as above does satisfy the 4 properties. Property 1 clearly holds since a hit of $(miss(g), j)$ means that node $miss(g)$ indeed has a value of j. Given the trial simulation and since everything is deterministic, $miss(g)$'s group leader will send a message in round r of $\mathcal{P}$'s execution. Furthermore, since the probes are done sequentially and since the player must have

encountered this hit before the adversary is ready to inject the next batch of failures, the adversary will not fail the dummy node connecting $miss(g)$'s group leader to the root at the beginning of round $r + 1$.

For Property 2, we need to prove that if a group g's group leader sends an influential message in round r, then all nodes in g have been hit by the end of round r. If $miss(g)$ does not exist, we already hit all nodes in g. If $miss(g)$ exists, let j be the value of the node $miss(g)$. Clearly, there has never been a probe $(miss(g), j)$. By our construction of the probing strategy in round r, $(miss(g), j)$ will be a candidate probe. If the player indeed probes $(miss(g), j)$ in round r, then $miss(g)$ will be hit in round r and we are done. If the player does not probe $(miss(g), j)$ in round r, the only possibility is that the adversary is ready to inject the next batch of failures at the beginning of round $r + 1$. In such a case, the adversary will fail the dummy node connecting g's group leader to the root, making the message (if any) sent by g's group leader non-influential.

For Property 3, if the adversary does not inject failures at the beginning of round $r+1$, then clearly the property inherited from the beginning of round r continues to hold at the beginning of round $r + 1$. If the adversary does inject failures at the beginning of round $r + 1$, then the group membership in round r and the group membership in round $r + 1$ are different. Let g_1, g_2, ..., g_l be the l groups in round r, immediately after the adversary potentially inject failures at the beginning of round r. Without loss of generality, assume that after the failures are injected, $g_{i+l/2}$ is merged with g_i (for $1 \leq i \leq l/2$) to form a new group, with g_i's leader being the leader of the new group. By inductive hypothesis on Property 3, immediately after the adversary potentially injects failures at the beginning of round r, $g_{i+l/2}$ $(1 \leq i \leq l/2)$ has at most one node that has not been hit. Given how the adversary injects failures, we know that the leader of g_i $(1 \leq i \leq l/2)$ must have sent an (influential) message in round r or earlier and after group g_i is formed. By Property 2 (both in round r and in earlier rounds), we know that all nodes in group g_i have been hit by the end of round r. This means that after merging g_i and $g_{i+l/2}$, the new group still only has at most one node that has not been hit.

Finally for Property 4, consider any given group g whose leader sends an influential message in round r. By Property 2, we know that all nodes in g have been hit by the end of round r, and thus the player knows all their values. Same as in the earlier

trial simulation, by inductive hypothesis on Property 4, the player can generate all influential messages sent by other group leaders up to and including round $r-1$. This means that the player can generate all incoming messages (up to and including round $r-1$) that may affect nodes in g. By same arguments as earlier, the player will be able to simulate $\mathcal{P}$'s execution on all nodes in g from round 0 to round r (inclusive). Also note that the messages received in round r, which we do not know yet, will not affect the messages sent by g's group leader in round r. Thus the player can generate that influential message sent by the group leader in round r. □

7.4 Lower Bound on the Number of Hits in the Probing Game

With the connection between SUM and the probing game proved in the previous section, we now intend to obtain a lower bound on the number of hits in the probing game. Such lower bound will later translate to a lower bound on the communication complexity of SUM. A simpler version of this probing game was analyzed in [25] to reason about silence-based communication, in a failure-free setting. There the player is not allowed to interleave probes on w_i with probes on $w_{i'}$. Thus for our purpose, we prove the following lower bound result on the probing game where the probes may be arbitrarily interleaved:

Lemma 7.4.1. *Consider the set U of all the $(3n)^n$ possible inputs to the probing game ($n \geq 2$) and any given (adaptive) deterministic probing strategy that can give correct (zero-error) results for at least $\frac{2}{3}$ fraction of those inputs. There must exist an input such that using that strategy, the player encounters n hits under that input.*

Proof. We prove by contradiction, and assume that there exists a deterministic probing strategy $\mathcal{S}$ that gives correct results for at least $\frac{2}{3}$ fraction of inputs and has at most $n-1$ hits for all inputs.

Consider any given input $W = (w_1, w_2, ..., w_n) \in U$, which is initially unknown to the player. At any point of time during the game, we define w_i's *residual domain*

(denoted as D_i) to be the set $\{j\}$ if there has been a probe (i, j) so far which is a hit. Otherwise w_i's *residual domain* D_i is defined to be the set:

$$\{0, 1, 2, ..., 3n - 1\} \setminus \{j \mid \text{there has been a probe } (i, j)\}$$

Intuitively, w_i's residual domain is the possible domain of w_i given the probe outcomes so far. When the game ends under $\mathcal{S}$, W has a unique *residual domain vector* $D = (D_1, D_2, ..., D_n)$, where D_i is the residual domain of w_i. We next prove a simple useful property on W's residual domain vector to facilitate later reasoning. We claim that for any input $Z = (z_1, z_2, ..., z_n)$ where $z_i \in D_i$, the probes done by the player and all the probe outcomes must be identical under input W and input Z. This in turn implies that Z will have the same residual domain vector as well as the same final output as that of W. We prove this claim for the kth probe, via a simple induction on k. The induction base for the zeroth probe clearly holds. Now consider the kth probe. We already know that all previous probes and their outcomes are identical under W and under Z. Since the player is deterministic, we know that the kth probe will be the same under W and Z. Let this probe be (i, j). If (i, j) is a hit for W, then we must have $D_i = \{j\}$. Since $z_i \in D_i$, we know that $z_i = j$ and the probe (i, j) will be a hit for Z as well. If (i, j) is a miss for W, then we must have $j \notin D_i$. Since $z_i \in D_i$, we know that $z_i \neq j$ and thus the probe will be a miss for Z as well. Thus the outcome of the kth probe will be identical under input W and input Z.

We now leverage the above property to prove the following claim. Define U' to be that set of inputs such that for each input $W \in U'$, when the game ends, in W's residual domain vector $D = (D_1, D_2, ..., D_n)$ there exists some D_i where $|D_i| \geq 2$. We claim that in order for the player to generate results correctly for at least $\frac{2}{3}$ fraction of all the inputs, $|U'|$ must be no larger than $\frac{2}{3}|U|$.

We prove the above claim by contradiction and assume that $|U'| > \frac{2}{3}|U|$. We partition U' into disjoint subsets such that all inputs in the same subset have the same residual domain vector. Consider any such subset U'_D where all inputs in the set has the same $D = (D_1, D_2, ..., D_n)$ as their residual domain vectors. By the earlier property on residual domain vector, we know that U'_D contains at least all those inputs Z where $z_i \in D_i$ for $1 \leq i \leq n$, and all such inputs Z will result in the same output. Furthermore, if an input Z' is such that $z'_i \notin D_i$ for some i, then it is impossible for Z' to be in U'_D. In other words, U'_D must be exactly the set of those inputs Z where

$z_i \in D_i$ for $1 \leq i \leq n$. Next without loss of generality, assume that $|D_1| \geq 2$. Consider any given $j_2 \in D_2$, $j_3 \in D_3$, ..., $j_n \in D_n$. For each $j \in D_1$, $Z = (j, j_2, j_3, ..., j_n)$ must be in U'_D, and the player will produce the same output. Such output can be correct for at most one j. This in turn means that the player can generate a correct result for at most $\frac{1}{2}$ fraction of the inputs in U'_D. Therefore for all the inputs in the set U', the player can generate correct results for at most $\frac{1}{2}$ fraction as well. Thus the player can generate a result correctly for at most $\frac{1}{2}|U'| + |U \setminus U'| = \frac{1}{2}|U'| + |U| - |U'| = |U| - \frac{1}{2}|U'| < \frac{2}{3}|U|$ inputs.

We have just proved that $|U \setminus U'| \geq \frac{1}{3}|U|$. We next use this result to prove that under some input in $U \setminus U'$, the number of hits will be n. For every input $W \in U \setminus U'$, we know that W's residual domain vector $D = (D_1, D_2, ..., D_n)$ satisfies $|D_i| = 1$ for all i. Essentially, the player has "pinpointed" the value of each w_i and knows W precisely, instead of only its sum. This means that in the probing game, the player actually needs to learn the input precisely for at least $\frac{1}{3}$ fraction of all the inputs. We next prove that for the player to achieve such a goal, there will be n hits on at least one of the inputs in $U \setminus U'$.

Given such a goal, consider any given point of time where the player decides to do a probe (i, j). (This necessarily means that there has not been a hit on w_i, and also means that $j \in D_i$ where D_i is current residual domain of w_i.) Since the goal is to learn the input precisely, doing such a probe is no different from doing any other probe (i, j') as long as $j' \in D_i$. With such an observation, we can now make the following without loss of generality assumption: For any given input W and any given i, consider all probes done by the player in the form of (i, j_0), (i, j_1), (i, j_2), ... Without loss of generality, we can assume that $j_0 = 0$, $j_1 = 1$, $j_2 = 2$, ... We know that under the probing strategy $\mathcal{S}$, there are at most $n - 1$ hits under all inputs, including all inputs $W \in U \setminus U'$. Consider any given $W \in U \setminus U'$ and let i be the index of the component that has not been hit. Since $|D_i| = 1$ and by our earlier without loss of generality assumption, we know that $w_i = 3n - 1$. However, the number of such inputs is:

$$|\{W \mid W \in U \text{ and } \exists i \text{ such that } w_i = 3n - 1\}| = (3n)^n - (3n-1)^n < \frac{1}{3}(3n)^n \text{ , for } n \geq 2.$$

This contradicts with $|U \setminus U'| \geq \frac{1}{3}|U|$. Thus under some $W \in U \setminus U'$, there will be n hits. □

7.5 Proof for Theorem 1.4.2

We are now ready to prove a series lemmas, which directly lead to Theorem 1.4.2. The following lemma is proved by first establishing a connection between randomized complexity and distributional complexity via well-known techniques [50], and then using Theorem 7.3.1 and Lemma 7.4.1 to obtain a lower bound on the distributional complexity. In the following, the notation $\mathcal{R}^{\text{syn,ft}}_{0,\delta}$ simply means $\mathcal{R}^{\text{syn,ft}}_{\epsilon,\delta}$ with $\epsilon = 0$.

Lemma 7.5.1. *Consider any $b \geq 1$ and any integer $N = \frac{(n+1)(n+2)}{2}$ where n is a power of 2. If in the* SUM *problem each node may take an integer value in $\{0, 1, ..., 3n-1\}$, then there exists a connected topology G with N nodes, such that:*

$$\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b) \quad \geq \quad \log n + 1$$

Proof. We let G be the topology constructed in Section 7.2, and consider the deterministic and adaptive adversary described there. It is easy to verify the number of edge failures is at most $2N$. For the sake of convenience, we view this adversary as part of the SUM protocol. Namely, given any randomized SUM protocol, we can consider a randomized "augmented protocol" which repeatedly executes one round of the randomized SUM protocol and then invokes the (deterministic) adversary to potentially inject failures. We thus no longer need to discuss the adversary separately.

Now consider the optimal randomized augmented protocol that generates a zero-error result with probability at least $\frac{2}{3}$ for all input W, while incurring a *worst-case* (over the coin flips) communication complexity of $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b)$. If we subject this protocol against an input chosen uniformly randomly out of all possible inputs, then trivially the protocol still generates a zero-error result with probability at least $\frac{2}{3}$, where the probability is taken over both the input distribution and the random coin flips. Now let us view this randomized augmented protocol as a distribution over deterministic augmented protocols. Then there must exist at least one deterministic protocol $\mathcal{P}$ which can generate a zero-error result with probability at least $\frac{2}{3}$ under this uniform input distribution, since otherwise the expectation taken over all deterministic augmented protocols cannot reach $\frac{2}{3}$. Finally, let a denote the

maximum number of bits sent by a node, across all nodes when we run $\mathcal{P}$ against the worst-case input (that maximizes a). Since $\mathcal{P}$ is selected by the randomized algorithm with positive probability and since $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b)$ is defined over worst-case coin flips, we have $a \leq \mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b)$.

On the other hand, Theorem 7.3.1 tells us that given such a $\mathcal{P}$, there exists a corresponding probing strategy $\mathcal{S}$ in the probing game so that using $\mathcal{S}$, the player in the probing game can generate the same result as $\mathcal{P}$. Since $\mathcal{P}$ generates a zero-error result for at least $\frac{2}{3}$ fraction of the inputs, we know that the player using $\mathcal{S}$ generates a zero-error result for so many inputs as well. Next by Lemma 7.4.1, there exists some input W such that the player encounters n hits. In turn, Theorem 7.3.1 now tells us that when running the SUM protocol $\mathcal{P}$ against this input W, some node sends at least $\log n + 1$ bits. Thus we have $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b) \geq a \geq \log n + 1$. □

The following corollary extends the above lemma to our standard setting where nodes only have binary values and also where N can be any integer.

Corollary 7.5.1. *Consider any $b \geq 1$ and any integer $N \geq 5$. Let n be the largest integer that is a power of 2 and satisfies $\frac{(n+1)(n+2)}{2} + n(3n-1) \leq N$. There exists a connected topology G with N nodes, such that:*

$$\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b) \quad \geq \quad \log n + 1$$

Proof. Let $N_1 = \frac{(n+1)(n+2)}{2}$ and we first construct a connected topology G_1 with N_1 nodes as described in Section 7.2. Next we attach $(3n-1)$ degree-1 *follower* nodes to each work node in G_1, and attach $(N - N_1 - n(3n-1))$ degree-1 nodes to the root of G_1. All those degree-1 nodes attached to G_1's root will always have value 0. Let G be the resulting N-node connected topology. One can trivially obtain a reduction from the SUM problem on G_1 (where each worker node has an integer value in $\{0, 1, ..., 3n-1\}$) to the SUM problem on G (where each node has a binary value). In particular in the reduction, the root in G_1 will simulate the root in G and also all the degree-1 neighbors of the root in G. Each worker node in G_1 will simulate the corresponding worker node and its $(3n-1)$ followers in G. If the worker node i in G_1 has a value of w_i, then in G the first w_i of the corresponding work node's follower nodes will have value 1 and the remaining $(3n-1-w_i)$ follower nodes will have

value 0. Combining this reduction with the lower bound on the SUM problem on G_1 from Lemma 7.5.1, we have $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G, 2N, b) \geq \log n + 1$. □

The next corollary extends the above corollary to $\mathcal{R}^{\text{syn,ft}}_{\epsilon,\frac{1}{3}}$ for $\epsilon \geq \frac{1}{N}$.

Corollary 7.5.2. *Consider any $b \geq 1$, any integer $N \geq 15$, and any $\epsilon \in [\frac{1}{N}, \frac{1}{15}]$. Let n be the largest integer that is a power of 2 and satisfies $\frac{(n+1)(n+2)}{2} + n(3n-1) \leq \frac{1}{3\epsilon}$. There exists a connected topology G with N nodes, such that:*

$$\mathcal{R}^{\text{syn,ft}}_{\epsilon,\frac{1}{3}}(\text{SUM}, G, 2N, b) \quad \geq \quad \log n + 1$$

Proof. Let $N_1 = \frac{1}{3\epsilon}$ and we first construct a connected topology G_1 with N_1 nodes such that $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G_1, 2N, b_1) \geq \log n + 1$ for any b_1. Corollary 7.5.1 ensures that such G_1 exists. Next we attach $N_2 = N - N_1$ nodes to the root of G_1, and let the resulting topology be G. Those N_2 nodes will always have a value of 0 and will never fail. Note that the final sum on G can never be above $\frac{1}{3\epsilon}$. If we have at most ϵ relative error on the final sum on G, then the absolute error is at most $\frac{1}{3\epsilon} \cdot \epsilon = \frac{1}{3}$. Since the exact sum must be an integer, to generate a result with ϵ relative error in G, the protocol intuitively needs to produce a zero-error result. Putting it another way, if the output is not an integer, we can always output the closest integer instead. Doing so will never cause an output that was previously within ϵ-error bound to exceed the ϵ-error bound after the conversion. The above observation enables a trivial reduction from $\mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G_1, 2N, b_1)$ to $\mathcal{R}^{\text{syn,ft}}_{\epsilon,\frac{1}{3}}(\text{SUM}, G, 2N, b)$, which gives:

$$\mathcal{R}^{\text{syn,ft}}_{\epsilon,\frac{1}{3}}(\text{SUM}, G, 2N, b) \;\geq\; \mathcal{R}^{\text{syn,ft}}_{0,\frac{1}{3}}(\text{SUM}, G_1, 2N, b_1) \;\geq\; \log n + 1$$

□

Combining Corollary 7.5.1 and Corollary 7.5.2 enables us to easily prove Theorem 1.4.2:

Proof for Theorem 1.4.2. We first prove the lower bound on $\mathcal{R}^{\text{syn,ft}}_{0}(\text{SUM}_N, b)$. For any $N \geq 5$, consider the N-node connected topology G as constructed by Corol-

lary 7.5.1. Together with Lemma 2.6.1, we trivially have:

$$\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, 2N, b) \geq \frac{1}{3}\mathcal{R}_{0,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) \geq \frac{1}{3}\mathcal{R}_{0,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}, G, 2N, b) \geq \frac{1}{3}(\log n + 1)$$

By Corollary 7.5.1, here n is the largest integer that is a power of 2 and satisfies $\frac{(n+1)(n+2)}{2} + n(3n-1) \leq N$. Thus we have $n = \Theta(\sqrt{N})$, which implies $\mathcal{R}_0^{\text{syn,ft}}(\text{SUM}_N, 2N, b) = \Omega(\log N)$.

We next prove the lower bound on $\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b)$ for $\epsilon \geq \frac{1}{N}$. For any $N \geq 15$, consider the N-node connected topology G as constructed by Corollary 7.5.2. We trivially have:

$$\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) \geq \mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}, G, 2N, b) \geq \log n + 1$$

By Corollary 7.5.2, here n is the largest integer that is a power of 2 and satisfies $\frac{(n+1)(n+2)}{2} + n(3n-1) \leq \frac{1}{3\epsilon}$. Thus we have $n = \Theta(\frac{1}{\sqrt{\epsilon}})$, which implies $\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) = \Omega(\log \frac{1}{\epsilon})$.

Finally, for $\epsilon = \Omega(\frac{1}{N})$ but $\epsilon < \frac{1}{N}$ (in which case ϵ is necessarily $\Theta(\frac{1}{N})$), we have:

$$\mathcal{R}_{\epsilon,\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) \geq \mathcal{R}_{\frac{1}{N},\frac{1}{3}}^{\text{syn,ft}}(\text{SUM}_N, 2N, b) = \Omega(\log N) = \Omega(\log \frac{1}{\epsilon})$$

□

Chapter 8

Upper Bound on the FT communication complexity of general CAAFs

This chapter proposes a novel fault-tolerant protocol for Sum. From the protocol, we show an upper bound on the FT communication complexity of Sum as following theorem:

Theorem 8.0.1. *For any $b \geq 21c$ and $1 \leq f \leq N$, $\mathcal{R}_0^{\text{syn,ft}}(\text{Sum}_N, f, b) = O((\frac{f}{b} + 1) \cdot \min(f \log N, \log^2 N))$.*

As can be seen later, same as some existing Sum protocols, our Sum protocol and its guarantees trivially generalized to arbitrary CAAFs as well. This gives an upper bound on general CAAFs as follow:

Corollary 1.4.2 (Restated)**.** *For any $b \geq 21c$ and $1 \leq f \leq N$,*

$$\mathcal{R}_0^{\text{syn,ft}}(\text{CAAF}_N, f, b) = O\left(\left(\frac{f}{b} + 1\right) \cdot \min\left(f \log N, \log^2 N\right)\right)$$

In the following, Section 8.1 first gives an overview of our fault-tolerant protocol for Sum. The protocol relies on two novel building blocks, i.e., Agg and Veri. Section 8.2 presents Agg and proves its properties while Section 8.3 is for Veri. Having

N	number of nodes in the topology G
b	SUM protocol's time complexity, in terms of flooding rounds
n	size of two-party problems
d	diameter of the topology G
f	upper bound on the number of edge failures
c	diameter of the topology never exceeds cd due to failures
t	parameter in AGG and VERI
l	a node's level in the aggregation tree

Table 8.1: Key notations in Chapter 8.

these properties, we will finally prove Theorem 8.0.1 in Section 8.4. Our protocol assumes that f is *known* to the protocol. Section 8.5 generalize our protocol to deal with *unknown* f.

8.1 Overview and Intuition

This section gives an overview of the structure of our upper bound protocol (Algorithm 3) that is used to prove Theorem 8.0.1. Our protocol relies on two novel building blocks:

- We first propose a novel deterministic aggregation protocol AGG (parameterized by $t \geq 0$), with time complexity of $O(1)$ flooding rounds and communication complexity of $O((t+1)\log N)$ bit. If the actual number of edge failures is no more than t, AGG always generates a correct result. Note that setting $t = f$ directly gives us $O(1)$ time complexity and $O(f \log N)$ communication complexity, which is already much better than the two existing SUM protocols mentioned earlier. The key technique in AGG is to take *speculative* actions to save time, instead of waiting for failures to be detected and then falling back to a second plan. We further carefully design a distributed mechanism to determine which speculative actions' effects should be retained or discarded, while using only local information.
- If the number of edge failures exceeds t, AGG may *unknowingly* generates a wrong result. We hence design a novel distributed verification protocol VERI, which aims to tell whether AGG's result is correct. VERI is also parameterized

by t and incurs $O(1)$ time complexity and $O((t+1)\log N)$ communication complexity. The key technique in Veri is that we allow it to have one-sided error. Specifically, we allow Veri to sometimes err when Agg does not err. Veri also employs a similar distributed mechanism to the one in Agg to avoid the need for global information in its execution.

Algorithm 3 Our upper bound protocol. Here b, c, and f are input parameters with $b \geq 21c$.

1: $x = \lfloor \frac{b-2c}{19c} \rfloor$; use public coins to select $\log N$ integers, with replacement, from the range of $[1, x]$; let the selected integers be $y_1, y_2, ..., y_{\log N}$, in non-decreasing order;
2: **for all** integer $i \in [1, \log N]$ **where** ($i = 1$ **or** $y_i \neq y_{i-1}$) **do**
3: sequentially invoke Agg and Veri, both with $t = \lfloor \frac{2f}{x} \rfloor$, from flooding round $((y_i - 1) \times 19c + 1)$ to $(y_i \times 19c)$;
4: **if** (Agg does not abort **and** Veri outputs `true`) **then** output Agg's result and terminate;
5: **end for**
6: invoke the existing brute-force Sum protocol in the last $2c$ flooding rounds, output its result, and terminate;

Here we give an overview of the structure of our upper bound protocol (Algorithm 3) that is used to prove Theorem 8.0.1. Given total b flooding rounds as a constraint on time complexity, we divide the first $b - 2c$ flooding rounds into $x = \Theta(b)$ *intervals*, with each interval having $\Theta(1)$ flooding rounds. Thanks to the small $O(1)$ time complexity of Agg and Veri, running Agg followed by Veri will take at most one interval. If the f edge failures were evenly distributed across all the intervals, then each interval would have $\frac{f}{x}$ edge failures. In such a case, running Agg parameterized with $t = \frac{f}{x}$ in any single interval would already produce a correct result, while incurring a desirable communication complexity of $O((\frac{f}{b} + 1)\log N)$. Here recall that t is the number of edge failures that Agg intends to tolerate, and the communication complexity of Agg is $O((t+1)\log N)$.

Since the edge failures are not always evenly distributed, we need a more complex design. Specifically, the nodes use public coins to select $\log N$ intervals uniformly randomly. In each selected interval, the nodes execute Agg and Veri sequentially, both with $t = \lfloor \frac{2f}{x} \rfloor$. One can easily see that with probability at least $\frac{1}{2}$, a random interval has no more than t edge failures. Hence with probability at least $1 - \frac{1}{N}$, the number of edge failures in *some* selected interval is small enough for Agg to tolerate. But if there have been more than t edge failures in an interval, then Agg

may *unknowingly* produce a wrong result. A difficulty here is that we cannot easily determine the number of edge failures that have occurred in a given interval, since it involves fault-tolerant counting while tolerating potential additional failures during counting. Hence instead of checking the number of edge failures in a given interval, our protocol invokes Veri after Agg, and then checks the condition at Line 4. If the condition is met, the protocol outputs Agg's result and terminates. By Theorem 8.2.3 and 8.3.2 later, such a result must be correct. Furthermore by Theorem 8.2.2 and 8.3.2 later, if the number of edge failures in an interval is no more than t, then the condition at Line 4 is guaranteed to be met.

Having given an intuitive overview on the protocol's correctness, we move on to look at its communication complexity. Since there can be at most f intervals with failures, Agg and Veri will be executed at most $\min(f+1, \log N) = O(\min(f, \log N))$ times. The communication complexity incurred by each Agg and Veri invocation is $O((t+1)\log N)$ bits, resulting in total $O((\frac{f}{b}+1)\min(f\log N, \log^2 N))$ bits in all the intervals. Next, the probability of reaching Line 6 is at most $\frac{1}{N}$. As explained in Section 1.3.1, the communication complexity of the brute-force Sum protocol is $O(N\log N)$. Hence the communication complexity incurred at Line 6, over average coin-flips, is $O(\log N)$.

The above intuitions eventually lead to Theorem 8.0.1, whose full proof is deferred to Section 8.4. Next in Section 8.2 and 8.3, we focus on Agg and Veri and prove their properties.

8.2 The Agg Protocol

Overview. Agg has an input parameter t ($t \geq 0$), which is the number of edge failures that it intends to tolerate. When running Agg, a node will flood[1] a special symbol to abort Agg once it has sent $(11t+14)(\log N+5)$ bits. Such an abort will never be triggered (as we prove later) if the actual number of edge failures is no larger than t. If the actual number of edge failures exceeds t, aborting before the communication complexity gets large enables Agg to properly bound its communication complexity.

[1]Throughout this chapter, a node *floods* a certain message by first sending the message to its neighbors, and then the other nodes simply forward that message upon first receiving it.

AGG first constructs a spanning tree and does a standard tree-based aggregation, where each non-root node sends its *partial sum* upstream along the tree. The *partial sum* of a node (either non-root or root) is the sum of the node's own input and all the partial sums received from its children. A key impact of failures is that they may block and prevent certain partial sums from propagating upstream. If a partial sum from a node B is blocked, a natural solution is to have B flood its partial sum, since flooding has the maximum resilience against failures. If the flooding does reach the root, the root can then incorporate B's partial sum to the final result. A second thought, however, shows that even with flooding, B's partial sum may still fail to reach the root if B's entire neighborhood fails immediately after B initiates the flooding. When this happens, the system needs to fall back and flood the partial sums of B's children, or B's descendants if B's children have also failed.

The key challenge here is that we need to do this within $O(1)$ flooding rounds. We cannot afford to wait to see whether B's partial sums get successfully flooded, and then fall back to flooding some other partial sums if things did not go well. To save time, we will have to do floodings *speculatively*, before knowing which floodings will be needed. This in turn leads to a second challenge: There will be overlap (or duplicates) in the partial sums received by the root (e.g., partial sums from both B and some of B's descendants). We need a careful mechanism to avoid double counting, which is non-trivial, especially without global knowledge about the tree topology.

The following sections present the details of AGG. At a high-level, AGG has 3 sequential phases: i) spanning tree construction and tree-aggregation (Section 8.2.1), ii) identifying potentially blocked partial sums and (speculatively) flooding them (Section 8.2.2), and iii) using a distributed mechanism based on *witnesses* to avoid double counting (Section 8.2.3). To facilitate understanding, the discussion in these 3 sections will be intuitive — we leave the pseudo-code to Section 8.2.4 and formal proofs to Section 8.2.5 and Section 8.2.6.

8.2.1 Tree Construction/Aggregation and Some Key Concepts

This section first describes the tree construction/aggregation phase in AGG, which is largely standard. Next we formalize a number of new concepts that are key for our later design.

Tree construction and aggregation. To construct the tree, the root first sends a tree_construct message. A node B waits for the first tree_construct message it receives. Let A denote the sender of that message. B sends an ack message indicating to A that B is A's child, and then sends a tree_construct message itself to continue constructing the tree. B's failing before sending ack will be equivalent to B not present in the network. The failure of B after sending ack will be dealt with later in Agg.

From now on in this chapter, the notions of "parent", "child", "ancestor", and "descendant" will always be with respect to this tree. Next Agg does standard tree-aggregation. Consider a given node B, and let l be its *level* (i.e., its distance from the root). Node B acts in the $(cd - l + 1)$th round during tree-aggregation, by summing up its own input with all the partial sums received from its children so far, and then sending the new partial sum to B's parent. Note that B does not necessarily wait for a message from each of its children, since some may have failed. Each partial sum thus is the sum of inputs from a subset of the nodes, and we also say that the partial sum *includes* those inputs.

Some key concepts. We say that a node B at level l experiences a *critical failure* if it fails after sending ack during tree construction and before taking its action in the $(cd - l + 1)$th round during tree-aggregation. Such a critical failure can be easily detected by B's parent A (if A is alive) during that round, when A does not receive the scheduled message from B. We want critical failure to become global knowledge when possible. To do so, A will flood a message claiming that B experiences a critical failure. We say that a flooding is *successful* if the flooded message eventually reaches the root. One can easily see that a successful flooding must reach all live nodes within cd rounds. We say that a critical failure is *visible* if it is eventually seen by the root. Otherwise it is *invisible*. To help understanding, the next will first assume that all critical failures are visible, and then remove that assumption in Section 8.2.6 (see Lemma 8.2.3).

Imagine that we remove all those edges connecting visible critical failures with their corresponding parents. Doing so partitions the aggregation tree into many smaller trees which we call *fragments* (Figure 8.1). A node's *local ancestors (descendants)* are all its ancestors (descendants) within the node's fragment. Each fragment also has its own *local root*. A fragment has a clean property: The partial sum of a node never includes inputs from nodes outside of its fragment, since those inputs have been

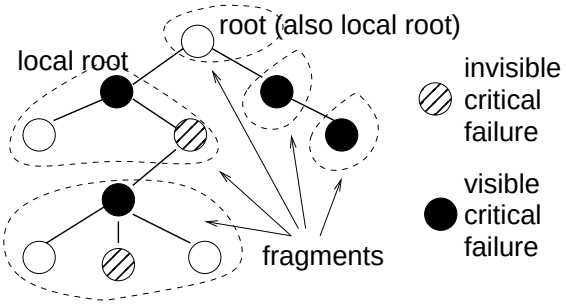

Figure 8.1: Example aggregation tree and fragments.

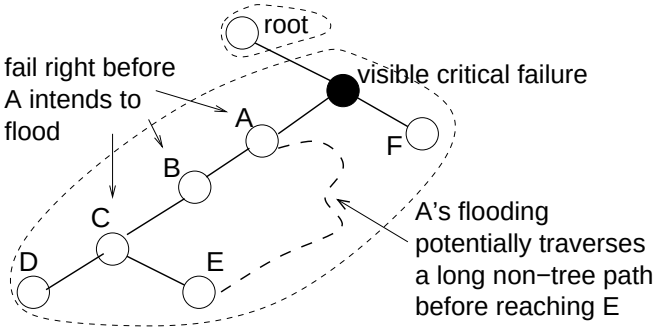

Figure 8.2: Why speculative flooding is needed.

blocked by the visible critical failures. Hence we can restrict most of our discussions to within a fragment.

A node A's partial sum is a *representative* of a node B iff i) A is either B itself or A is B's local ancestor, and ii) the tree path from A to B (excluding A and B) contains no invisible critical failures. Intuitively, B's representative must include B's input. A *representative set* is a set of partial sums with the following property: For any node B, if B is alive at (has failed by) the end of the VERI execution that immediately follows AGG, then a representative set contains exactly one (at most one) representative of B. Intuitively, if we obtain a representative set and sum up all the partial sums there, we get a correct sum result.

8.2.2 Identify and Flood Potentially Blocked Partial Sums

With the above notion of representative set, our goal in the remainder of AGG is for the root to obtain a representative set. If there were no critical failures at all, then

the root's partial sum by itself is already a representative set. With critical failures, a representative set will contain not only the root's partial sum but also those blocked partial sums. Consider the example in Figure 8.2. Here, the root's partial sum, A's partial sum, and F's partial sum form a representative set. Imagine that we have A and F flood their partial sums, so that the root can get those and add those to the final result. However, A, B, and C all fail right before A intends to flood. Hence A's partial sum is lost and we now need D and E to flood their partial sums, which will form a second representative set together with the root's and F's partial sum. Ideally, D and E should do so after they know that A's flooding has failed. Unfortunately, it can take one flooding round before such determination can be made, since A's flooding could traverse a long non-tree path before reaching E (Figure 8.2). Similarly, if E's flooding also fails, then E's local descendants (if any) may need to wait one more flooding round before taking action.

This example shows that to have small time complexity, nodes need to flood speculatively, before knowing that the flooding is needed. AGG uses the following elegant design to decide which node initiates flooding at what time: The root always floods its partial sum in the first round of the partial sum flooding phase. A non-root node B at level l floods its partial sum in the $(l+1)$th round of the phase, iff in that round it does not receive any flooding message (containing any partial sums) from its parent A. Here A may or may not be the *initiator* of the corresponding flooding. This design has two important features. First, the design never does excessive floodings: If A has not failed by the $(l+1)$th round, B must receive some message from A and will not initiate its own flooding. This implies that the total number of floodings is linear with the number of edge failures. Second, Lemma 8.2.4 in Section 8.2.6 proves that the design always floods a superset of those partial sums that need to be flooded. Namely, if B did not flood its own partial sum, then B must have forwarded a "better" partial sum that includes all those inputs included by B's partial sum.

8.2.3 Avoid Double Counting While Using Only Limited Information

The root potentially receives many flooded partial sums, and it needs to pick a representative set to avoid double counting. The partial sums seen by the root can be

classified into three mutually-exclusive categories: The partial sum of a node B is *dominated* if the root also sees A's partial sum where A is B's local ancestor. A non-dominated partial sum of a node B is *compulsory* if either B or at least one of B's local descendants is still alive at the end of the VERI execution that immediately follows AGG, otherwise the non-dominated partial sum is called *optional*. Lemma 8.2.5 in Section 8.2.6 proves that the union of all compulsory partial sums and any subset of optional partial sums forms a representative set.

Without knowledge of the global tree topology for labeling each partial sum, AGG maintains distributed topology information to do so. Specifically, when initially constructing the tree, AGG lets each node learn the ids of its nearest $2t$ ancestors. Interestingly, such limited information is already sufficient for AGG to select a representative set, in the following way via *witnesses*.

Having witnesses label partial sums. A node B's *witness* is either B itself or some local descendant of B whose distance to B is at most t. Let the local root of B's fragment be X and consider B's witness C. First, if C sees X among its $2t$ ancestors, then its $2t$ ancestor must contain all of B's local ancestors. Note that partial sums seen by the root must be seen by all live nodes as well. Hence if C sees a partial sum from some local ancestor of B's, C knows that B's partial sum is dominated and will thus flood its determination $\langle \mathsf{dominated}, B \rangle$ to inform the root. Otherwise C floods its determination $\langle \mathsf{compulsory||optional}, B \rangle$. When B has multiple witnesses, such determination may be flooded multiple times. This does not increase communication complexity since all the determinations are identical, and a node only needs to participate in one such flooding.

Second, if C does not see X among its $2t$ ancestors, then there must be at least $2t - t = t$ nodes on the tree path from B to X (excluding B and X). If the number of edge failures is no more than t, then there must be at least one live node on that tree path between B and X. That live node must have successfully flooded a partial sum of either itself or one of its local ancestors. This implies that B's partial sum must be dominated. C will thus flood the determination $\langle \mathsf{dominated}, B \rangle$. If the number of edge failures exceeds t, such determination might be wrong, which will be dealt with later by VERI.

Finally, it is possible for all of B's witnesses to fail (or for their floodings to fail to reach the root). In such a case, B's farthest local descendant must be no more than

t hops away from B, since otherwise the number of edge failures will be more than t. (Again, the case where the number of edge failures exceeds t will be dealt with by VERI.) This implies that B and its local descendants must have all failed, since they are all B's witnesses. Hence if the root does not receive any determination on B's partial sum, the root knows that the partial sum cannot be compulsory, and must be either dominated or optional.

Take all three cases into account and by Lemma 8.2.5 discussed earlier, to form a representative set, the root simply includes in the set a partial sum from a node B iff $\langle \mathsf{compulsory}||\mathsf{optional}, B\rangle$ has been received.

8.2.4 Pseudo-Code for The AGG Protocol

Algorithm 4 presents the pseudo-code for the AGG protocol. Following are some additional comments on the pseudo-code. By default, the sender of a message always attaches its id on the message (not shown in the pseudo-code), allowing the receiver to infer the sender. A "_" field in a received message means that we do not care about the value of that field. The pseudo-code allows a node to send multiple messages in a single round. In actual implementation, all these messages should be combined into one, and can thus be sent in one round. The pseudo-code invokes the **flood** primitive in several places, whose (trivial) implementation is not included in the pseudo-code. For a node to flood a message, the node sends the message to its neighbors. Any node receiving a flooded message simply forward that message upon first receiving that message. The initiating node is called the *source* of the flooding. Note that if a node receives a second flooded message (potentially initiated by a different source) with the same context, the node will *not* forward it again. Finally, each node in AGG keeps track of the total number of bits it has sent. Once the number reaches $(11t + 14)(\log N + 5)$, a node will flood a special symbol to cause all nodes to abort AGG. This mechanism is not shown in the pseudo-code, for clarity.

8.2.5 Time Complexity and Communication Complexity of AGG

Following theorem summarizes the time and communication complexity of AGG

Algorithm 4 The AGG Protocol

/* Tree Construction Phase (total $2cd + 1$ **rounds) */**
if (I am the root) **then**
  $level = 0$; $parent =$ `null`; $ancestor[i] =$ `null` for all $i \in [1, 2t]$; $children = \emptyset$;
  send $\langle$tree_construct, $level$, $ancestor\rangle$ in round 1;
else
  wait to receive the first message (with arbitrary tie breaking if multiple messages received in the same round) in the form of $\langle$tree_construct, $sender\_level$, $sender\_ancestor\ \rangle$ from any node u;
  let the current round be r; $level = sender\_level + 1$; $parent = u$; $children = \emptyset$;
  $ancestor[1] = parent$; $ancestor[i] = sender\_ancestor[i-1]$ for all $i \in [2, 2t]$;
  send $\langle$ack, $parent\rangle$ in round r; send $\langle$tree_construct, $level$, $ancestor\rangle$ in round $r + 1$;
end if
upon receiving message in the form of $\langle$ack, $my\_id\rangle$ from any node v: $children = children \bigcup \{v\}$;
/* Aggregation Phase (total $2cd + 1$ **rounds) */**
$psum = my\_input$; $max\_level = level$; // $psum$ is for "partial sum"
for all $v \in children$ **do**
  if (in round $cd - level + 1$ of this phase, message in the form of $\langle$aggregation, $sender\_psum$, $sender\_max\_level\rangle$ from node v is received) **then**
    $psum = psum + sender\_psum$; $max\_level = \max(max\_level, sender\_max\_level)$;
  else
    flood $\langle$critical_failure, $v\rangle$ in round $cd - level + 1$ of this phase;
  end if
end for
send $\langle$aggregation, $psum$, $max\_level\rangle$ in round $cd - level + 1$ of this phase;
/* Speculative Flooding Phase (total $2cd + 1$ **rounds) */**
if (I am the root) **then flood** $\langle$flooded_psum, my_id, $psum\rangle$ in round 1 of this phase;
if (I am not the root **and** no message from $parent$ is received in round $level + 1$ of this phase) **then**
  flood $\langle$flooded_psum, my_id, $psum\rangle$ in round $level + 1$ of this phase;
end if
/* Partial Sum Selection Phase (total $cd + 1$ **rounds) */**
$ancestor[0] = my\_id$;
for all message received in the form of $\langle$flooded_psum, $source\_id$, _$\rangle$ **do**
  let $i \in [0, 2t]$ be the smallest i such that $ancestor[i] = source\_id$; let $i = \infty$ if such i does not exist;
  let $j \in [0, 2t]$ be the smallest j such that $ancestor[j]$ is the root or $\langle$critical_failure, $ancestor[j]\rangle$ has been received; let $j = \infty$ if such j does not exist;
  $dom =$ I have received a message $\langle$flooded_psum, $ancestor[k]$, _$\rangle$ with $k \in [i + 1, j]$;
  if $(i \le t)$ **and** $(i \le j)$ **then** // I am a witness
    If $(j = \infty)$ **then flood** $\langle$dominated, $source\_id\rangle$ in round 1 of this phase;
    If $(j \ne \infty$ **and** $dom)$ **then flood** $\langle$dominated, $source\_id\rangle$ in round 1 of this phase;
    If $(j \ne \infty$ **and** $(!dom))$ **then flood** $\langle$compulsory||optional, $source\_id\rangle$ in round 1 of this phase;
  end if
end for
/* Output Phase (only executed by the root) */
$sum = 0$;
for all received message in the form of $\langle$flooded_psum, $source\_id$, $source\_psum\rangle$ **do**
  if ($\langle$compulsory||optional, $source\_id\rangle$ has been received) **then** $sum = sum + source\_psum$;
  // messages $\langle$dominated, $source\_id\rangle$ are not actually needed, and we sent those only for clarity
end for
output sum;

Theorem 8.2.1. *The time complexity and communication complexity of* Agg *are no more than* $11c$ *flooding rounds and* $O((t+1)\log N)$ *bits, respectively.*

Proof. The pseudo-code in Algorithm 4 obviously shows that Agg terminates within $7cd + 4$ rounds, which are at most $11c$ flooding rounds. For communication complexity, recall that in Agg, a node will flood a special symbol to abort Agg once it has sent $(11t + 14)(\log N + 5)$ bits. Hence the communication complexity is $O((t + 1)\log N)$. □

8.2.6 Correctness Properties of Agg

This section will prove following two theorems which summarize the correctness property of Agg:

Theorem 8.2.2. *If there are at most t edge failures during the execution of* Agg*, then* Agg *never aborts and always outputs a correct result.*

Theorem 8.2.3. *If there is no LFC, then* Agg *either outputs a correct result or aborts.*

Here the concept of LFC is defined as follow:

Definition 8.2.1 (Long failure chain (LFC))**.** *With respect to a pair of* Agg *and* Veri *execution (both with parameter t), a* long failure chain *(LFC) is a chain of t nodes* $A_1, A_2, ..., A_t$ *within the same fragment such that i)* A_i *is the parent of* A_{i+1} $(1 \leq i \leq t-1)$*, ii) all of them have failed by the end of the* Agg *execution, and iii)* A_t *has at least one local descendant that is alive at the end of the* Veri *execution. Here the notions of fragment, parent, and etc are all defined based on the* Agg *execution.* A_1 *and* A_t *are called the* head *and* tail *of the LFC, respectively.*

Note that having no more than t edge failures implies no LFC, while the reverse is not true. Theorem 8.2.3 claims that regardless of the number of edge failures, Agg will not err as long as there is no LFC.

Throughout this section, unless otherwise mentioned, *nodes on a tree path* from node A to node B includes all nodes on the path as well as the two end points A and B. All "Phases" and "Lines" in the proofs, by default, refer to phases and lines in Algorithm 4.

Lemma 8.2.1. *At the end of the Tree Construction Phase in* Agg*, there exists a distributed* aggregation *tree in the system where each node on the tree knows its children, parent, and* $2t$ *ancestors. Furthermore, if a node in the system is not included in this aggregation tree, then it must have failed by the end of the Tree Construction Phase.*

Proof. Trivial from the pseudo-code. □

From now on, whenever we refer to a node, by default we mean a node in the above aggregation tree. By the above lemma, nodes not on the tree must have failed by the end of Agg and hence their inputs do not need to be included in the sum result.

Lemma 8.2.2. *Consider any flooding done in any phase in the* Agg *protocol. If the flooded message is initiated or received by a node that is still alive at the end of the phase, then all nodes that are still alive at the end of the phase will have received the message by the end of the phase.*

Proof. In Agg, flooded messages are initiated at Line 18, 25, 34, 35, and 36. One can easily verify that in all cases, there are at least $cd+1$ rounds (including the round during which the flood is initiated) remaining in the corresponding phase. Within those $cd+1$ rounds, such flooding is either seen by all live nodes, or is completely smothered by failures and does not reach any of the remaining live nodes. But since the message is initiated or received by a node that is still alive at the end of the phase, it is impossible for the flooding to be completely smothered. □

Lemma 8.2.3. *If Z is an invisible critical failure, then all of Z's local ancestors must have failed by the end of the Aggregation Phase in* Agg*.*

Proof. Prove by contradiction and assume that Z's local ancestor A is still alive. Let Y be the node with the smallest level on the tree path from Z to A such that Y is a critical failure. In fact in this case, Y must be an invisible critical failure. Y can be Z itself, but Y must not be A since A is still alive. In the next we will prove that Y's parent will initiate a flooding claiming that Y is a critical failure, and this flooded message will successfully reach A. Since A is alive even at the end of the Aggregation Phase, by Lemma 8.2.2, this flooding will be forwarded by A and

eventually reach the root. This will imply that Y is a visible critical failure instead of an invisible one, leading to a contradiction.

To see why Y's parent will initiate a flooding and why such flooding will successfully reach A, consider any give node B on the tree path from Y's parent to A. Let l be B's level. By definition of Y, B must not be a critical failure, and hence B is alive during round $cd - l + 1$ at Line 21. Hence Y's parent will initiate a flooding of message $\langle \mathsf{critical\_failure}, Y \rangle$ at Line 18. Furthermore, this flooded message will be properly relayed by every node on the path from Y's parent to A. □

The next lemma shows that our design on when to do speculative floodings has the following nice property: If a live node B does not flood its own partial sum, then it must have forwarded a "better" partial sum that includes all those inputs included by B's partial sum. In other words, we never run into the situation where we need B's partial sum but B did not flood it.

Lemma 8.2.4. *Consider any node B that is alive by the end of the Speculative Flooding Phase of* Agg *and whose level is l. Then in round $(l + 1)$ of the Speculative Flooding Phase, B must either flood its own partial sum at Line 25 or forward a partial sum (of its local ancestor) that includes all those inputs included by B's partial sum.*

Proof. We will prove, via a simple induction, that if B doesn't flood its own partial sum then B must have forwarded a partial sum of one of its local ancestors. By definition of a local ancestor and by Lemma 8.2.3, we know that such a partial sum includes all those inputs included by B's partial sum.

Let l be B's level. The induction base for $l = 0$ is trivial. Assume that our claim holds for $l = k - 1$, and consider the node B at level k. If B does not flood its own partial at Line 25 of the Agg protocol, then B must have received a message contains some partial sums from its parent A at Line 24. Since B is alive by the end of the Speculative Flooding Phase, B must not be a critical failure and hence A must be B's local ancestor. Also, all of A's local ancestors must be B's local ancestors. If the message contains A's partial sum, we are done. If the source of this flooded message is not A, by inductive hypothesis, A must have forwarded (in the message)

a partial sum of one of A's local ancestors. Since A's local ancestors must be B's local ancestors, we are done as well. □

Lemma 8.2.5. *The union of all compulsory partial sums and any subset of optional partial sums must form a representative set.*

Proof. Let the given union be S. We need to prove that for any node B, if B is alive at (has failed by) the end of the VERI execution that immediately follows AGG, then S contains exactly one (at most one) representative of B. We first prove that S contains at most one representative of B, via a contradiction. A representative of B is either B's partial sum or B's local ancestor's partial sum. Hence if S contains two partial sums s_1 and s_2 that are both representatives of B, then one of s_1 and s_2 must be dominated. This contradicts to the fact that S contains no dominated partial sums.

We next prove that if B is alive at the end of the VERI execution that immediately follows AGG, then S contains at least one representative of B. By Lemma 8.2.4, B must either flood its own partial or forward a partial sum that includes B's input. In either case, the partial sum flooded or forwarded is B's representative. Since B is alive at the end of the VERI execution that immediately follows AGG, by Lemma 8.2.2, this partial sum will be received by the root. Now consider the set containing all of B's representatives that are received by the root. This set is hence non-empty. There must be at least one partial sum in this set that is non-dominated. We claim that this non-dominated partial sum must be compulsory. To see why, note that this partial sum must be from either B or B's local ancestor. Since B is alive at the end of the VERI execution that immediately follows AGG, this non-dominated partial sum must be compulsory. By definition of S, this compulsory partial sum must be in S. □

The next lemma proves that if there is no LFC, then the labels (i.e., "compulsory||optional" and "dominated") assigned by the witnesses on the partial sums are always correct:

Lemma 8.2.6. *Consider all partial sums received by the root at Line 41. If there is no LFC, then at Line 42:*

- *For every dominated partial sum from a node B, the root does not receive* ⟨compulsory||optional, B⟩.

- *For every compulsory partial sum from a node B, the root receives* $\langle$compulsory||optional, $B\rangle$.

Proof. We prove the two cases one by one:

- Prove by contradiction, and assume that the root receives $\langle$compulsory||optional, $B\rangle$ flooded by a node C. By Line 33, C is at most t hops away from B, and C must be either B's local descendant or B itself. Since B's partial sum s_1 is dominated, then there must exists another partial sum s_2 (seen by the root and hence all nodes in the system) that is from B's local ancestor A. If C does not see the local root of the fragment among its $2t$ ancestors, C will have $j = \infty$ at Line 31 and thus will not flood $\langle$compulsory||optional, $B\rangle$ at Line 36. If C sees the local root among its $2t$ ancestors, C must also see A among its local ancestors. This means that the *dom* variable at Line 32 is `true`, and hence C will not flood $\langle$compulsory||optional, $B\rangle$ at Line 36 either. Contradiction.

- We first claim that there exists a node C that is still alive at the end of the Agg and C satisfies the conditions at Line 33 (i.e., C must be B's witness). To see why, note that since B's partial sum is compulsory, B must have a local descendant D that is still alive at the end of the corresponding Veri execution. Now consider the tree path from D to B. There must be a node C that is within t hops of B and that is still alive at the end of Agg, since otherwise together with the existence of D, we would have an LFC. Such a C obviously satisfies the conditions at Line 33.

  Next we prove, via a contradiction, that C must see the local root of C's fragment among C's $2t$ ancestors. If C does not see the local root, then there are at least $2t - t = t$ nodes on the tree path from B to the local root (excluding B and the local root). Since B's partial sum is compulsory, B must have a local descendant D that is still alive at the end of the corresponding Veri execution. Now we can claim that there must exists a node A on the tree path from B to the local root (excluding B and the local root) that is still alive at the end of Agg. This is true because otherwise we would have an LFC. Next by Lemma 8.2.4, A must have flooded its own partial sum or a partial sum of one of its local ancestors. By Lemma 8.2.2, the flooded partial sum will be received

by the root in time. Since this partial sum is from B's local ancestor, it means that B's partial sum is dominated, leading to a contradiction.

Hence C must see its local root within its $2t$ ancestors and C will have $j \neq \infty$ at Line 31. Since B's partial sum is compulsory, the root (and C as well) must have not seen another partial sum from one of B's local ancestors. This means that the *dom* variable at Line 32 is `false` for C. Now C has satisfied the conditions at Line 36, and thus will flood $\langle$compulsory||optional, $B\rangle$. Finally, since C is still alive at the end of the Agg, by Lemma 8.2.2, such flooding will reach the root.

□

Theorem 8.2.3 (Restated)**.** *If there is no LFC, then* Agg *either outputs a correct result or aborts.*

Proof. We prove that if there is no LFC and if Agg does not abort, then it outputs a correct sum. Agg computes a final output by summing up all *source_psum*'s in messages $\langle$flooded_psum, *source_id*, *source_psum*$\rangle$ (Line 41) where $\langle$compulsory||optional, *source_id*$\rangle$ has been received (Line 42). By Lemma 8.2.5 and 8.2.6, all these *source_psum*'s exactly form a representative set S. Each partial sum in S is the sum of the inputs from some of the nodes. By definition of a representative set, for any node B that is still alive at (has failed by) the end of the Veri execution that immediately follows Agg, S must contain exactly one (at most one) partial sum (i.e., B's representative) that includes the input of B. Hence the sum of all the partial sums in S includes B's input exactly once if B is still alive at the end of the Veri execution that immediately follows Agg, or at most once if B has failed. Finally, the sum of all the partial sums in S obviously does not include the input of any nodes that are not in the aggregation tree. By Lemma 8.2.1, nodes that are not on the aggregation tree must have failed by the end of the Tree Construction Phase and hence there is no need to include their inputs. All these imply that the sum result must be correct. □

Theorem 8.2.2 (Restated)**.** *If there are at most t edge failures during the execution of* Agg*, then* Agg *never aborts and always outputs a correct result.*

Proof. No more than t edge failures implies no LFC. Hence by Theorem 8.2.3, it suffices to prove that no node sends $(11t + 14)(\log N + 5)$ bits to abort Agg. Algorithm 4 shows that in Agg a node may i) send messages at Line 9, 9, and 21, and ii) initiate floodings at Line 18, 23, 25, 34, 35, and 36. One can easily verify that Line 9, 9, and 21 incur at most $10 + 2\log N$, $10 + 2\log N + 2t\log N$, and $10 + 3\log N$ bits respectively. Note that here the 10 bits are sufficient to encode the type of each message. Also, each message needs to reserve $\log N$ bits for the sender's id.

Next because there are at most t edge failures, the total sizes of all the messages flooded at Line 18, 23, and 25 are $t(10 + 2\log N)$, $10 + 3\log N$, and $t(10 + 3\log N)$ bits, respectively. Finally, at Line 34, 35, and 36, a node may flood either $\langle$dominated, *source_id*$\rangle$ or $\langle$compulsory||optional, *source_id*$\rangle$ for each received $\langle$flooded_psum, *source_id*, *source_psum*$\rangle$. Because the number of distinct *source_id* is at most $t + 1$, the number of flooded messages with distinct contents will be at most $2t + 2$. Since each such message has no more than $10 + 2\log N$ bits, all those floodings at Line 34, 35, and 36 incur at most $(2t + 2)(10 + 2\log N)$ bits for each node. Adding all these numbers up yields exactly $60 + 40t + 14\log N + 11t\log N$ bits, which is less than $(11t + 14)(\log N + 5)$ bits. □

8.3 The Veri Protocol

Overview. Veri aims to determine whether Agg's output is correct. The natural approach is for Veri to determine whether there have been more than t edge failures. This turns out to be difficult since it involves fault-tolerant counting while tolerating potential additional failures during counting. Instead, our approach is to i) identify a weaker requirement that is nevertheless sufficient for Agg not to err, and ii) allow Veri to sometimes err when Agg does not err. Such a weaker requirement on Veri eventually makes an efficient design possible.

Specifically, with respect to a pair of Agg and Veri execution (both with parameter t), a *long failure chain* (LFC) is a chain of t nodes $A_1, A_2, ..., A_t$ within the same fragment such that i) A_i is the parent of A_{i+1} $(1 \le i \le t-1)$, ii) all of them have failed by the end of the Agg execution, and iii) A_t has at least one local descendant that is alive at the end of the Veri execution. Here the notions of fragment, parent, and etc

scenario	AGG	VERI
1. no more than t edge failures (implying no LFC)	output correct result	output `true`
2. more than t edge failures and no LFC	output correct result or abort	no guarantee
3. more than t edge failures and exists LFC	no guarantee	output `false`

Table 8.2: Guarantees of AGG and VERI under different scenarios.

are all defined based on the AGG execution. A_1 and A_t are called the *head* and *tail* of the LFC, respectively. Note that having no more than t edge failures implies no LFC, while the reverse is not true. Theorem 8.2.3 claims that regardless of the number of edge failures, AGG will not err as long as there is no LFC.

Theorem 8.2.3 (Restated)**.** *If there is no LFC, then* AGG *either outputs a correct result or aborts.*

The theorem implies that VERI may safely err in the 2nd scenario in Table 8.2, where there are more than t edge failures but no LFC. Table 8.2 also summarizes the guarantees of AGG and VERI in all other possible scenarios.

Next, section 8.3.1 focuses on intuitions of our design. Pseudo-code is presented in and we leave formal proof to Section 8.3.3 and Section 8.3.4.

8.3.1 Design of The VERI Protocol

Recall that VERI aims to determine whether AGG's output is correct. The natural approach is for VERI to determine whether there have been more than t edge failures. This turns out to be difficult since it involves fault-tolerant counting while tolerating potential additional failures during counting. Instead, our approach is to i) identify a weaker requirement (i.e., no LFC exists) that is nevertheless sufficient for AGG not to err, and ii) allow VERI to sometimes err when AGG does not err. Such a weaker requirement on VERI eventually makes an efficient design possible.

By the above discussion, we design VERI by focusing on detecting LFCs. Similar to AGG, in VERI once a node has sent $(5t + 7)(3 \log N + 10)$ bits, it will flood a special symbol to cause VERI to output `false`.

Strawman design assuming no additional failures. To help understanding, we first describe a strawman design while assuming that there are no additional failures occurring during VERI's execution. A simple way to detect LFCs is for each node to ping its parent and children on the (aggregation) tree, and to flood the information about detected failures to all other nodes. Those *failed parents* and *failed children* are potentially tails and heads of LFCs. Without knowing the global tree topology, we will leverage the same witnesses as in Section 8.2.3 to determine whether they are indeed tails and heads of LFCs. Consider a failed parent B and a witness C of B's. Recall that B's witness is either B itself or some local descendant of B whose distance to B is at most t. C finds, among its $2t$ ancestors, B's nearest ancestor A such that A is either a failed child or a fragment boundary. One can easily see that B is the tail of an LFC iff A is at least $t-1$ hops away from B. Thus C can precisely determine whether B is the tail of an LFC, and can flood such determination to inform the root.

Failures of the witnesses. We now move on to the actual VERI design, by explaining how different kinds of failures during VERI's execution are addressed. We first consider the failures of the witnesses: In the earlier example, it is possible for all of B's witnesses to fail, so that no node can make a proper determination. We overcome this key challenge precisely by allowing VERI to err, as explained below.

First, we need AGG to maintain some additional information: During AGG's aggregation phase, we have each node learn the maximum level among its local descendants. This can be easily done by having nodes propagate upstream, along with the partial sum, the maximum level it has seen among its local descendants. Now in VERI, imagine that we can infer the distance x from B to B's farthest local descendants.[2] If the root does not receive any determination on whether B is the tail of some LFC (implying that all of B's witnesses have failed), the root applies the following rule: If $x \leq t$, it claims that B is not the tail of an LFC. Otherwise it claims that B is the tail of an LFC, and outputs `false`.

[2]Since B may have failed early on, we may not be able to actually get x. Nevertheless, one can achieve a similar functionality by using the maximum level information from B's descendants. See Section 8.3.4 for details.

To see when the above rule gives a correct/wrong determination, we separately consider two cases. First, $x \leq t$ implies that all of B's local descendants are B's witnesses. They must have all failed since all witnesses have failed. In turn, by definition B must not be the tail of an LFC. Second, $x > t$ implies that B has at least t witnesses and all of them have failed. We still cannot determine whether there exists an LFC. But since VERI is allowed to make one-sided error when there are more than t edge failures (i.e., the 2nd and 3rd scenario in Table 8.2), VERI can simply output `false` in such a case.

When to detect failures. We move on to consider additional failures during the detection of failed parents/children. Those failures may prevent the floodings of information about failed parents/children from reaching the root. This is similar to flooded partial sums getting lost in Section 8.2.2 and Figure 8.2. To deal with this, VERI uses the following elegant design similar to the one in AGG: The root floods a single bit. If a node at level l does not receive this bit or any message (claiming the detection of failed parents) from its own parent within $l + 1$ rounds, it floods a message claiming that its own parent is a failed parent. If B is the tail of an LFC, such design guarantees (Lemma 8.3.2 in Section 8.3.4) to inform the root that either B or some of B's local descendant is a failed parent.

Detection of failed children is similarly done by propagating a single bit upstream along all the tree edges. Finally, VERI always detects failed parents first and then detects failed children. This is necessary for correctness, if additional failures may occur during VERI. We leave the details on how this ordering is leveraged in our proofs to Section 8.3.4.

8.3.2 Pseudo-Code for The VERI Protocol

Algorithm 5 presents the pseudo-code for the VERI protocol. All the additional comments in Section 8.2.4 about Algorithm 4 apply to Algorithm 5 as well, except the following: In VERI, once a node has sent $(5t + 7)(10 + 3 \log N)$ bits, it will flood a special symbol to terminate VERI and cause the root to output `false`.

It is worth noting that VERI detects failed parents first, and then failed children. This

Algorithm 5 The VERI Protocol. The initial values of the variables *parent*, *children*, *ancestor*, *level*, and *max_level* are all from the previous AGG execution.

```
1: /* Failed Parent Detection Phase (total 2cd + 1 rounds) */
2: if (I am the root) then
3:     flood ⟨detect_failed_parent⟩ in round 1;
4: else
5:     if (no message from parent is received in round level + 1) then
6:         flood ⟨failed_parent, parent, max_level − level + 1⟩ in round level + 1;
7:     end if
8: end if

9: /* Failed Child Detection Phase (total 2cd + 1 rounds) */
10: if (children = ∅) then // I am a leaf
11:     flood ⟨detect_failed_child⟩ in round cd − level + 1 of this phase;
12: else
13:     for all node v ∈ children do
14:         if (no message from node v is received in round cd − level + 1 of this phase) then
15:             flood ⟨failed_child, v⟩ in round cd − level + 1 of this phase;
16:         end if
17:     end for
18: end if

19: /* LFC Detection Phase (total cd + 1 rounds) */
20: for all received messages in the form of ⟨failed_parent, v, _⟩ do
21:     let i ∈ [0, 2t] be the smallest i such that ancestor[i] = v; let i = ∞ if such i does not exist;
22:     let j ∈ [0, 2t] be the smallest j such that ancestor[j] is either the root or ⟨critical_failure, ancestor[j]⟩
        was previously received in AGG; let j = ∞ if such j does not exist;
23:     if (i ≤ t and i ≤ j) then // I am a witness
24:         let k ∈ [i, 2t] be the smallest k such that i) ⟨failed_child, ancestor[k]⟩ has been received, or ii)
            ancestor[k] is the root, or iii) ⟨critical_failure, ancestor[k]⟩ was previously received in AGG;
            let k = ∞ if such k does not exist;
25:         if (k − i + 1 ≥ t) then
26:             flood ⟨LFC_tail, v⟩ in round 1 of this phase;
27:         else
28:             flood ⟨not_LFC_tail, v⟩ in round 1 of this phase;
29:         end if
30:     end if
31: end for

32: /* Output Phase (only executed by the root) */
33: if (I have received message ⟨LFC_tail, v⟩ for any node v) then output false; // LFC exists
34: for all received message in the form of ⟨failed_parent, v, x⟩ where x ≥ t do
35:     if (⟨not_LFC_tail, v⟩ has not been received) then output false; // LFC may exist — VERI may have
        one-sided error here
36: end for
37: output true; // no LFC
```

ordering is intentional and is necessary for correctness – the proofs for Theorem 8.3.3 and Lemma 8.3.4 rely on such ordering.

8.3.3 Time Complexity and Communication Complexity of VERI

Theorem 8.3.1. *The time complexity and communication complexity of* VERI *are no more than* $8c$ *flooding rounds and* $O((t+1)\log N)$ *bits, respectively.*

Proof. The pseudo-code in Algorithm 5 clearly shows that VERI always terminates within $5cd + 3$ rounds, which are at most $8c$ flooding rounds. For communication complexity, recall that in VERI, a node will flood a special symbol to terminate VERI once it has sent over $(5t+7)(10+3\log N)$ bits. Hence the communication complexity is $O((t+1)\log N)$. □

8.3.4 Correctness Properties of VERI

This section will prove following theorem which suberizes the correctness property of VERI:

Theorem 8.3.2. *Consider a pair of* AGG *and* VERI *execution, both parameterized by* t*. If there exists an LFC, then* VERI *must output* `false`*. If there are at most* t *edge failures, then* VERI *must output* `true`*.*

Throughout this section, we use *X.level* and *X.max_level* to denote the value of the local variables *level* and *max_level* on node X at the end of the AGG execution, respectively. Same to Section 8.2.6, whenever we refer to a node in this section, by default we mean a node in the aggregation tree as constructed in the AGG execution. Also *nodes on a tree path* from node A to node B, by default, includes all nodes on the path as well as the two end points A and B. All "Phases" and "Lines" in the proofs, by default, refer to phases and lines in Algorithm 5.

Lemma 8.3.1. *Consider any flooding done in any phase in the* VERI *protocol. If the flooded message is initiated or received by a node that is still alive at the end of the phase, then all nodes that are still alive at the end of the phase will receive the message by the end of the phase.*

Proof. In VERI, floodings are potentially initiated at Line 3, 6, 11, 15, 26, and 28. One can easily verify that in all cases, there are at least $cd + 1$ rounds (including the round during which the flooding is initiated) remaining in the corresponding phase. Within those $cd + 1$ rounds, such flooding is either seen by all live nodes, or is completely smothered by failures and does not reach any of the remaining live nodes. But since the message is initiated or received by a node that is still alive at the end of the phase, it is impossible for the flooding to be completely smothered. □

The next lemma formalizes the property of the Failed Parent Detection Phase. The lemma shows if a node B has failed and if it has a live descendant F, then the protocol is guaranteed to find a failed parent C on the tree path between B and F. (Note that the protocol does not necessarily find B itself as a failed parent, due to additional failures during VERI's execution.)

Lemma 8.3.2. *Consider a node B and any of its local descendant F. If B failed before the Failed Parent Detection Phase starts and if F is still alive at the end of that phase, then there must exist a node C such that: i) C is on the tree path from F to B, ii) all nodes on the tree path from C to B have failed by the end of the phase, and iii) every node that is alive at the end of the phase has received the message $\langle$failed_parent, C, $x\rangle$ by the end of that phase with $x \geq F.level - C.level$.*

Proof. Let E be the node with the smallest level on the tree path from F to B, such that E is still alive at the end of the phase. Since F is alive, such E must exist. Let C be the node on the tree path from E to B with the largest level that did not send the message which it is supposed to send in round $C.level + 1$. (Note that this intended message can either be a new flooding initiated by C itself at Line 6 or it can be a message received from C's parent and then forwarded by C.) C must exist since at least B, which failed before the phase starts, did not send the message. C already satisfies the first two properties needed in the lemma. The next proves that C satisfy the last property as well.

Let D be C's child that on the tree path from E to B. D must exist since C cannot be E which is alive at the end of the phase. Since C did not send any message in round $C.level + 1$, D will flood $\langle$failed_parent, C, $x\rangle$ where $x = D.max\_level - D.level + 1$ at Line 6. By definition of C, all nodes on the tree path from E to D manage to send

the messages that they are supposed to send during the corresponding rounds. Hence the message ⟨failed_parent, C, x⟩ will reach E. Finally, because E is alive at the end of the phase, Lemma 8.3.1 tells us that the every node that is alive by the end of the phase will receive ⟨failed_parent, C, x⟩.

We still need to show $x \geq F.level - C.level$. By the definition of C, D is still alive at the end of the previous Agg execution. By Lemma 8.2.3, there are no (invisible) critical failures on the tree path from F to D. This implies that $D.max\_level \geq F.max\_level \geq F.level$, and $x = D.max\_level - D.level + 1 \geq F.level - D.level + 1 = F.level - C.level$. □

Next we use the above lemma to prove the following theorem:

Theorem 8.3.3. *If there exists an LFC,* Veri *must output* `false`.

Proof. Let A and B be the head and tail of the given LFC, respectively. By definition of LFC, B has a local descendant F that is still alive at the end of Veri. By Lemma 8.3.2, there exist a node C on the tree path from F to B such that i) all nodes on the tree path from C to B have failed by the end of that phase, and ii) every node that is alive at the end of the Failed Parent Detection Phase receives ⟨failed_parent, C, x⟩ by the end of that phase where $x \geq F.level - C.level$.

We first claim that the root *may* receive the message ⟨LFC_tail, C⟩ but will never receive the message ⟨not_LFC_tail, C⟩, as proved in the following. For the message ⟨failed_parent, C, x⟩, consider any node D that satisfies Line 23 (i.e., D is C's witness). Since all nodes on the tree path from C to A have failed by the end of the Failed Parent Detection Phase, none of those nodes will flood ⟨failed_child, _⟩ at Line 15.[3] There are at least t nodes on the tree path from C to A. Thus at Line 25, D will have $k - i + 1 \geq t$ and hence D will never flood ⟨not_LFC_tail, C⟩ at Line 28.

Now if the root does receive the message ⟨LFC_tail, C⟩, then it will output `false` at Line 33 and we are done. Next consider the case where the root does not receive ⟨LFC_tail, C⟩. We claim that this implies that all of C's witnesses have failed by the end of the LFC Detection Phase. Prove by contradiction and assume that C's

[3] Note that the argument here relies on the fact that the Failed Parent Detection Phase is before the Failed Child Detection Phase.

witness D is still alive. D must have received $\langle$failed_parent, C, $x\rangle$ with $x \geq F.level - C.level$. Since D is a witness, it must satisfy the conditions at Line 23. It will either executed Line 26 or 28. By arguments in the previous paragraph, D will not flood $\langle$not_LFC_tail, $C\rangle$ and hence D must flood $\langle$LFC_tail, $C\rangle$ at Line 26. Since D is still alive at the end of the LFC Detection Phase, Lemma 8.3.1 tells us that the root will receive this message flooded by D, leading to a contradiction. Because F is still alive and is C's local descendant, and since all of C's witnesses have failed, it implies that $F.level - C.level \geq t + 1$ and thus $x \geq t + 1$. Thus the message $\langle$failed_parent, C, $x\rangle$ must satisfy the condition of $x \geq t$ at Line 34. Finally, since the root never receives $\langle$not_LFC_tail, $C\rangle$ by our earlier argument, it will output `false` at Line 35. □

The next lemma formalizes the property of the Failed Child Detection Phase. The lemma shows if a node D has failed, then unless all nodes from D to its local root have failed, the protocol is guaranteed to find a failed child C on the tree path between D and its local root. (Note that the protocol does not necessarily find D itself as a failed child, due to additional failures during VERI's execution.)

Lemma 8.3.3. *For any node D that failed before the Failed Child Detection Phase starts, there must exist a node C such that: i) C is on the tree path from D to D's local root, ii) all nodes on the tree path from D to C have failed by the end of the phase, and iii) either C is D's local root or every node that is alive at the end of the phase receives $\langle$failed_child, $C\rangle$ by the end of the phase.*

Proof. If all nodes on the tree path from D to its local root has failed by the end of the phase, the lemma trivially hold with C being the local root. Otherwise let A be the node with the largest level on the tree path from D to its local root, such that A is still alive at the end of the phase.

Let C be the node on the tree path from D to A with the smallest level that did not send the message which it is supposed to send in round $cd - C.level + 1$. (Note that this intended message can either be a new flooding initiated by C itself at Line 15 or it can be some message received from C's children and then forwarded by C.) C must exist since at least D, which failed before the phase starts, will not send the message. C already satisfies the first two properties needed in the lemma. The next proves that C satisfy the last property as well.

Let B be C's parent. B is on the tree path from D to A since C cannot be A which is alive at the end of the phase. Since C did not send any message in round $cd - C.level + 1$, B will flood $\langle \mathsf{failed\_child}, C \rangle$ at Line 15. By definition of C, all nodes on the tree path from B to A manage to send the messages that they are supposed to send at the corresponding rounds. Hence the message $\langle \mathsf{failed\_child}, C \rangle$ will reach A. Finally, because A is alive at the end of the phase, Lemma 8.3.1 tells us that the every node that is alive at the end of the phase will receive $\langle \mathsf{failed\_child}, C \rangle$. □

Leveraging the above lemma, we can now prove the following lemma. This lemma claims that if there are no more than t edge failures, then no node will ever flood $\langle \mathsf{LFC\_tail}, \_ \rangle$, and some node may flood $\langle \mathsf{not\_LFC\_tail}, \_ \rangle$.

Lemma 8.3.4. *Consider any pair of* Agg *and* Veri *executions during which the total number of edge failures is no more that t. For any node D such that the root has received* $\langle \mathsf{failed\_parent}, D, \_ \rangle$ *by the end of the Failed Parent Detection Phase, no node will ever flood* $\langle \mathsf{LFC\_tail}, D \rangle$. *Furthermore, if a witness of D is still alive at the end of the* Veri *execution, then that witness will flood* $\langle \mathsf{not\_LFC\_tail}, D \rangle$ *at Line 28.*

Proof. For the root to receive $\langle \mathsf{failed\_parent}, D, \_ \rangle$, some node must have flooded this message earlier at Line 6. For such flooding to be initiated, the condition at Line 5 must be met, implying that D has failed before the Failed Child Detection Phase.[4] Lemma 8.3.3 tells us that there exists node C on the tree path from D to its local root such that i) all nodes on the tree path from D to C have failed by the end of the Failed Child Detection Phase, and ii) either C is D's local root or $\langle \mathsf{failed\_child}, C \rangle$ is received by all nodes which is alive at the end of the Failed Child Detection Phase. Since C has failed by the end of the phase, it must not be the root and hence it has a parent.

If all of D's witnesses failed before the Failed Child Detection Phase starts, the lemma trivially holds. Otherwise let E be any witness of D's which is still alive at the beginning of the Failed Child Detection Phase. E cannot be D since D failed before the phase starts, and thus D has at least one child. Together with the earlier fact that C has a parent and the given condition that there are no more than t edge

[4] Note that the argument here relies on the fact that the Failed Parent Detection Phase is before the Failed Child Detection Phase.

failures, this implies that C is at most $t-2$ hops away from D. Finally, recall that either C is the D's local root (and hence E's local root) or the message $\langle$failed_child, $C\rangle$ is received by E. Hence at Line 24, E will find a k such that $k-i \leq t-2$. Such a value of k does not satisfy the condition at Line 25, preventing E from flooding $\langle$LFC_tail, $D\rangle$. In fact with such a value of k, if E is alive at the end of the VERI execution, E must flood $\langle$not_LFC_tail, $D\rangle$ at Line 28. □

Next we use the above lemma to prove the following theorem:

Theorem 8.3.4. *If there are no more than t total edge failures (during the executions of* AGG *and* VERI*), then* VERI *must output* `true`.

Proof. Prove by contradiction and assume that VERI outputs `false`. VERI may output `false` only in three cases. The first case is at Line 33, where the root receives $\langle$LFC_tail, $D\rangle$ for some node D. For the root to receive this message, there must have been some node E that floods $\langle$LFC_tail, $D\rangle$ at Line 26. For E to do so, it must see the message $\langle$failed_parent, D, _$\rangle$, which must also be seen by the root. Apply Lemma 8.3.4 and we know that no node will ever flood $\langle$LFC_tail, $D\rangle$. This contradicts with the fact that the root later receives this message.

The second case where VERI outputs `false` is at Line 35. This means that the root receives a message $\langle$failed_parent, D, $x\rangle$ with $x \geq t$, and it does not receive any message $\langle$not_LFC_tail, $D\rangle$. Let D's child E be the node that initially flooded the message $\langle$failed_parent, D, $x\rangle$ at Line 6. Hence $x = E.max\_level - E.level + 1 \geq t$. Thus E has at least $t-1$ local descendants, and in turn D has at least $t+1$ witnesses (i.e., D, E, and E's nearest $t-1$ local descendants). Since there are at most t edge failures, D must have at least one witness C that is still alive at the end of the VERI execution. Lemma 8.3.4 then tells us that C will flood $\langle$not_LFC_tail, $D\rangle$, and Lemma 8.3.1 tells us that such flooding will reach all live nodes. This contradicts with the fact that the root does not receive $\langle$not_LFC_tail, $D\rangle$.

The last case where VERI outputs `false` is when some node has sent $(5t+7)(10+3\log N)$ bits and hence floods a special symbol to terminate VERI. We will show that this will not happen, by carefully count the total number of bits sent by each node. In VERI, nodes only communicate by floodings. A node may initiate floodings at

Line 3, 6, 11, 15, 26, and 28. The size of the flooded messages is always no larger than $(10 + 3 \log N)$. Here 10 bits is sufficient to encode the message type. Also, each message needs to allocate $\log N$ bits for the sender's id. At each line except Line 11, the total number of floodings initiated system-wide is at most $(t + 1)$. At Line 11, each leaf initiates a flooding for the message ⟨detect_failed_child⟩. By our design, since all these floodings have the same content, a node will only forward the first such message received. Thus this is equivalent to a single flooding, in terms of the number of bits sent by each node. Taking all the above into account, a node will send at most $(5t + 6)(10 + 3 \log N)$ bits, which is less than $(5t + 7)(10 + 3 \log N)$ bits. □

Directly combining Theorem 8.3.3 and 8.3.4, we have:

Theorem 8.3.2 (Restated)**.** *Consider a pair of* Agg *and* Veri *execution, both parameterized by t. If there exists an LFC, then* Veri *must output* `false`*. If there are at most t edge failures, then* Veri *must output* `true`*.*

8.4 Proof for Theorem 8.0.1

Theorem 8.0.1 (Restated)**.** *For any $b \geq 21c$ and $1 \leq f \leq N$, $\mathcal{R}_0^{\text{syn,ft}}(\text{Sum}_N, f, b) = O((\frac{f}{b} + 1) \cdot \min(f \log N, \log^2 N))$.*

Proof. : We prove the theorem by constructing an upper bound protocol (Algorithm 3). For any given $b \geq 21c$, we divide the first $b - 2c$ flooding rounds into $x = \lfloor \frac{b-2c}{19c} \rfloor = \Theta(b)$ *intervals*, with each interval having at least $19c$ flooding rounds. The nodes use public coins to select $\log N$ intervals uniformly randomly (with replacement) out of all the intervals. Within each selected interval, the nodes execute Agg and Veri (both parameterized with $t = \lfloor \frac{2f}{x} \rfloor$) sequentially. If Agg does not abort *and* Veri outputs `true`, the protocol terminates and outputs the result generated by Agg. If the protocol does not output within the first $b - 2c$ flooding rounds, the root will flood a single bit to all nodes, taking c flooding rounds. After receiving this bit, all nodes will invoke the brute-force protocol for computing sum, in the last c flooding rounds. In this brute-force protocol, each node floods its id and its input to all other

nodes. Within c flooding rounds, the root is guaranteed to receive all flooded messages initiated by nodes that are still alive at the end of the protocol. The root then adds up the input for each id, and outputs the sum.

We first prove that the above protocol always produces a correct sum result. If the protocol outputs a sum by invoking the brute-force protocol, the result is trivially correct. If the protocol outputs the result generated by Agg, then we know that Agg does not abort *and* Veri outputs `true`. By Theorem 8.3.2, we know that there must have been no LFC. In turn by Theorem 8.2.3, we know that the result generated by Agg (if it did not abort) must be correct.

We move on to prove that the communication complexity of our upper bound protocol is $O((\frac{f}{b}+1)\cdot\min(f\log N,\log^2 N))$. By Theorem 8.2.2 and 8.3.2, if there are no more than $t=\lfloor\frac{2f}{x}\rfloor$ edge failures within an interval, then Agg will not abort and Veri will output `true`. This will then allow the upper bound protocol to terminate immediately after that interval. Since there are no more than f edge failures and since we selected $\log N$ intervals, we know that Agg and Veri will be executed at most $\min(f+1,\log N)$ times. By Theorem 8.2.1 and 8.3.1, the communication complexity of Agg and Veri are both $O((t+1)\log N)$. Hence the total communication complexity in all the intervals is $O((t+1)\cdot\min(f\log N,\log^2 N))$.

We next aim to reason about the communication complexity in the last $2c$ flooding rounds, where the brute-force protocol is invoked if no results have been generated so far. Since there are at most f edge failures in all the x intervals, with probability at least $\frac{1}{2}$, a uniformly random interval contains no more than $\lfloor\frac{2f}{x}\rfloor$ edge failures. Hence with probability at least $\frac{1}{2}$, by Theorem 8.2.2 and 8.3.2, Agg will not abort and Veri will output `true`, causing the upper bound protocol to terminate. The probability of invoking the brute-force protocol is thus at most $\frac{1}{2^{\log N}}=\frac{1}{N}$. The brute-force protocol itself has a communication complexity of $O(N\log N)$, implying that the communication complexity (over average-case coin flips) incurred in the last $2c$ flooding rounds is at most $O(\frac{1}{N}\cdot N\log N)=O(\log N)$.

Putting everything together, the communication complexity of the upper bound protocol is $O((t+1)\cdot\min(f\log N,\log^2 N))+O(\log N)=O((\frac{f}{b}+1)\cdot\min(f\log N,\log^2 N))$. □

8.5 Dealing with Unknown f

Our upper bound of $O((\frac{f}{b}+1)\cdot\min(f\log N, \log^2 N))$ in Theorem 8.0.1 assumes that f (i.e., the upper bound on the number of edge failures) is known to the protocol. It is trivial to generalize our upper bound protocol (in the proof of Theorem 8.0.1) to deal with unknown f, using the standard doubling trick.

Specifically, given b flooding rounds where $b \geq 19c\log N + 21c$, we divide the first $b-2c$ flooding rounds into $1+\log N$ *blocks*. In the ith block, our guess for f will be $f = 2^{i-1}$. Each block is further divided into $x = \lfloor \frac{b-2c}{19c(1+\log N)} \rfloor = \Theta(\frac{b}{\log N})$ *intervals*. Within the ith block, the nodes uniformly randomly select $\log N$ intervals. In each selected interval, we again run Agg and Veri, with $t = \lfloor \frac{2\cdot 2^{i-1}}{x} \rfloor$. As before, if Agg does not abort *and* Veri outputs `true`, the protocol terminates. Finally, if the protocol does not terminate within the first $b-2c$ flooding rounds, we again resort to the brute-force protocol.

The correctness of the above generalized protocol is obvious. Next we show that the communication complexity of the protocol is $O((\frac{f}{b}+1)\cdot\min(f\log^2 N, \log^3 N))$, which is still within polylog factor from our lower bound. For blocks 1 through $\lceil\log f\rceil+1$, by same argument as earlier, the total communication complexity incurred is:

$$\begin{aligned}
&\sum_{i=1}^{\lceil\log f\rceil+1} O\left(\left(\left\lfloor\frac{2\cdot 2^{i-1}}{x}\right\rfloor+1\right)\cdot\min(f\log N,\log^2 N)\right)\\
=\;& O\left(\left(\frac{f}{x}+\log f+1\right)\cdot\min(f\log N,\log^2 N)\right)
\end{aligned}$$

Next in block $\lceil\log f\rceil+1$ and later blocks, our guess on f (i.e., 2^{i-1}) already reaches the actual f. By same argument as earlier, the protocol will terminate in each of these blocks independently with at least $1-\frac{1}{N}$ probability. Hence the communication complexity incurred in block $\lceil\log f\rceil+2$ and later blocks is at most:

$$\begin{aligned}
&\sum_{i=\lceil\log f\rceil+2}^{\log N+1} \frac{1}{N^{i-\lceil\log f\rceil-1}}\cdot O\left(\left(\left\lfloor\frac{2\cdot 2^{i-1}}{x}\right\rfloor+1\right)\cdot\min(f\log N,\log^2 N)\right)\\
=\;& O\left(\frac{1}{N}\cdot\left(\frac{f}{x}+1\right)\cdot\min(f\log N,\log^2 N)\right)
\end{aligned}$$

Finally, the probability of the protocol of reaching the last $2c$ flooding rounds is

at most $\frac{1}{N}$. Hence the communication complexity incurred in the last $2c$ flooding rounds is at most $O(\frac{1}{N} \cdot N \log N) = O(\log N)$. Adding the three part up and we get the communication complexity of the protocol as:

$$\begin{aligned} & O\left(\left(\frac{f}{x} + \log f + 1 + \frac{1}{N} \cdot \left(\frac{f}{x} + 1\right)\right) \cdot \min(f \log N, \log^2 N)\right) + O(\log N) \\ = \; & O\left(\left(\frac{f}{b} + 1\right) \cdot \min(f \log^2 N, \log^3 N)\right) \end{aligned}$$

Chapter 9

Conclusions and Future Work

Tolerating failures has been a key focus of distributed computing research from the very beginning. Adding this fault tolerance requirement to multi-party communication complexity leads to the following natural question: "If we want to compute a certain function while tolerating a certain number of failures, what will the communication complexity be?" This thesis centers on above question, specifically on i) tolerating node *crash* failures, and ii) computing the function over *general* topologies.

This thesis has made following contribution: 1) We've shown an exponential gap between the non-fault-tolerant and fault-tolerant communication complexity of SUM; 2) We've proved near-optimal lower and upper bound on the fault-tolerant communication complexity of general commutative and associative aggregates; 3) We've introduced UNIONSIZECP, a new two-party problem comes with a novel *cycle promise*. We've further shown that such a problem not only enables our lower bounds on the fault-tolerant communication complexity of SUM, but also plays a fundamental role in reasoning about fault-tolerant communication complexity of many functions beyond SUM.

There are many interesting follow-up open questions on the subject:

- This thesis has proved a series of lower bounds for fault-tolerant communication complexity for SUM. Can we further strengthen our lower bounds? Note

that even our randomized (ϵ, δ)-approximate lower bound on the communication complexity of UnionSizeCP is not tight (i.e., roughly $\frac{1}{q}$ factor from the upper bound), and thus improvement might be possible even there.

- This thesis has focused on *oblivious* adversary, i.e., failure adversaries adversarially decide beforehand (i.e., before the protocol flips any coins) which nodes fail at what time. It is also meaningful to consider *adaptive* adversaries which decide during the execution. Our lower bounds can trivially extend to adaptive adversaries while our upper bound protocol can not. Can we find out a protocol for adaptive failure adversaries?

- This thesis has assumed that a node can send infinity number of bits in a single round. Many researches has considered the model that a node can send $O(\log N)$ bits in one round. Can we obtain interesting results in this setting?

For answering these questions, we believe that some of the insights developed in this thesis (e.g., on the role of failures in the reduction and on the cycle promise) can be valuable.

Bibliography

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, May 1996. 6, 24

[2] O. Ayaso, D. Shah, and M. Dahleh. Information theoretic bounds for distributed computation over networks of point-to-point channels. *IEEE Transactions on Information Theory*, 56(12):6020–6039, December 2010. 8

[3] T.C. Aysal, M.E. Yildiz, A.D. Sarwate, and A. Scaglione. Broadcast gossip algorithms for consensus. *IEEE Transactions on Signal Processing*, 57(7):2748–2761, July 2009. 6

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, June 2004. 9, 19, 43, 50, 51, 52, 53, 54

[5] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. *Journal of Computer and System Sciences*, 73(3):245–264, May 2007. 6, 12, 17

[6] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991. 7

[7] Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography*, March 2006. 7

[8] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of cryptography*, March 2008. 7

[9] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC*, May 1993. 7

[10] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, May 1988. 7

[11] P. Berman and J. Garay. Fast consensus in networks of bounded degree. *Distributed Computing*, 7(2):67–73, December 1993. 8

[12] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, June 2006. 6

[13] M. Braverman and A. Rao. Towards coding for maximum errors in interactive communication. In *STOC*, June 2011. 8

[14] J. Brody and A. Chakrabarti. A multi-round communication lower bound for gap hamming and some consequences. In *CCC*, July 2009. 10

[15] A. Calderbank, P. Frankl, R. Graham, W. Li, and L. Shepp. The Sperner capacity of linear and nonlinear codes for the cyclic triangle. *Journal of Algebraic Combinatorics*, 2(1):31–48, March 1993. 48

[16] A. Chakrabarti and O. Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. Technical report, 2010. 10

[17] A. Chandra, M. Furst, and R. Lipton. Multi-party protocols. In *STOC*, April 1983. 3, 8

[18] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *STOC*, May 1988. 7

[19] D. Chaum, I. Damgård, and J. Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, August 1987. 7

[20] Binbin Chen, Haifeng Yu, Yuda Zhao, and Phillip B. Gibbons. The Cost of Fault Tolerance in Multi-Party Communication Complexity. In *PODC*, July 2012. 3

[21] J. Chen and G. Pandurangan. Optimal gossip-based aggregate computation. In *SPAA*, June 2010. 6

[22] J. Chen, G. Pandurangan, and D. Xu. Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis. In *IPSN*, April 2005. 6

[23] B. Chlebus, D. Kowalski, and M. Strojnowski. Fast scalable deterministic consensus for crash failures. In *PODC*, August 2009. 8

[24] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, March 2004. 6, 12

[25] A. Dhulipala, C. Fragouli, and A. Orlitsky. Silence-based communication. *IEEE Transactions on Information Theory*, 56(1):350–366, January 2010. 91

[26] R. Duan and S. Pettie. Connectivity oracles for failure prone graphs. In *STOC*, June 2010. 16

[27] I. Eyal, I. Keidar, and R. Rom. LiMoSense — live monitoring in dynamic sensor networks. In *ALGOSENSORS*, September 2011. 6

[28] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, September 1985. 6

[29] M. Franklin and M. Yung. Communication complexity of secure computation. In *STOC*, May 1992. 7

[30] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *CRYPTO*, August 1987. 7

[31] L. Gargano and A. Rescigno. Communication complexity of fault-tolerant information diffusion. In *IEEE Symposium on Parallel and Distributed Processing*, December 1993. 8

[32] S. Gilbert and D. Kowalski. Distributed agreement with optimal communication complexity. In *SODA*, January 2010. 8

[33] A. Giridhar and P. Kumar. Towards a theory of in-network computation in wireless sensor networks. *IEEE Communications Magazine*, 44(4):98–107, April 2006. 8

[34] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, May 1987. 7

[35] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997. 4

[36] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In *Advances in Cryptology - ASIACRYPT*, December 2000. 7

[37] M. Hirt and J. Nielsen. Robust multiparty computation with linear communication complexity. In *CRYPTO*, August 2006. 7

[38] R. Impagliazzo and R. Williams. Communication complexity with synchronized clocks. In *CCC*, June 2010. 8, 19, 21, 24

[39] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005. 6

[40] P. Jesus, C. Baquero, and P. Almeida. Fault-tolerant aggregation by flow updating. In *DAIS*, June 2009. 6

[41] Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer, 2001. 9

[42] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan. Efficient gossip-based aggregate computation. In *PODS*, June 2006. 6

[43] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, October 2003. 6

[44] V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *ICDCN*, 2010. 8

[45] V. King and J. Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In *DISC*, 2009. 8

[46] V. King and J. Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine agreement with an Adaptive Adversary. In *PODC*, July 2010. 8

[47] V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *SODA*, 2006. 8

[48] V. King, J. Saia, V. Sanwalani, and E. Vee. Towards secure and scalable computation in peer-to-peer networks. In *FOCS*, 2006. 8

[49] F. Kuhn and R. Oshman. Dynamic networks: models and algorithms. *SIGACT News*, 42(1):82–96, March 2011. 15

[50] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996. 2, 18, 19, 20, 47, 48, 50, 94

[51] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, December 2002. 4

[52] M. McGlynn and S. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *MobiHoc*, October 2001. 3

[53] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *PODC*, July 2006. 6, 12

[54] S. Nath, P. Gibbons, S. Seshany, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. *ACM Transactions on Sensor Networks*, 4(2), March 2008. 6, 12

[55] I. Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, July 1991. 48, 52, 57

[56] R. Pietro and P. Michiardi. Brief announcement: Gossip-based aggregate computation: Computing faster with non address-oblivious schemes. In *PODC*, August 2008. 8

[57] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC*, May 1989. 7

[58] S. Rajagopalan and L. Schulman. A coding theorem for distributed computation. In *STOC*, May 1994. 8

[59] A. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. In *STOC*, June 2011. 34

[60] L. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. 8

[61] D. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, January 2004. 10, 29, 31

[62] A. Yao. Some complexity questions related to distributive computing. In *STOC*, April 1979. 2

[63] A. Yao. Protocols for secure computations. In *FOCS*, November 1982. 7

[64] A. Yao. How to generate and exchange secrets. In *FOCS*, October 1986. 7

[65] H. Yu. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. *Distributed Computing*, 23(5):373–394, April 2011. 6, 12

[66] Yuda Zhao, Haifeng Yu, and Binbin Chen. Near-optimal communication-time tradeoff in fault-tolerant computation of aggregate functions. In *PODC*, July 2014. 18