5-1-2015

Selfishness and Malice in Distributed Systems

George Saad

George Saad

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Prof. Jared Saia , Chairperson

Prof. Maxwell Young

Prof. Thomas Hayes

Prof. Dorian Arnold

Selfishness and Malice in Distributed Systems

by

George Saad

B.S., Computer Science, Alexandria University, 2004
M.S., Computer Science, Alexandria University, 2007
M.S., Computer Science, University of New Mexico, 2012

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May 2015

Dedication

To my parents, for their support and encouragement they're giving me for graduation.

– George Saad

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Jared Saia, for his excellent guidance, caring and patience, and for patiently correcting my writing and financially supporting my research.

Also, I would like to give many thanks to Dr. George Luger who gave me the first push to pursue my PhD at the University of New Mexico. It is hard to articulate how he has encouraged, supported, helped and given me excellent suggestions.

Further, I had several valuable discussions with Dr. Jedidiah Crandall and Dr. Dorian Arnold concerning network and system issues that are related to the dissertation. They supported me with helpful suggestions and recommendations.

Finally, I would like to mention that I would never have been able to finish my dissertation without the guidance of my committee members: Dr. Jared Saia, Dr. Maxwell Young, Dr. Thomas Hayes and Dr. Dorian Arnold.

Selfishness and Malice in Distributed Systems

by

George Saad

B.S., Computer Science, Alexandria University, 2004

M.S., Computer Science, Alexandria University, 2007

M.S., Computer Science, University of New Mexico, 2012

Ph.D., Computer Science, University of New Mexico, 2015

Abstract

Large-scale distributed systems are increasingly prevalent. Two issues can impact the performance of such systems: selfishness and malice. Selfish players can reduce social welfare of games, and malicious nodes can disrupt networks. In this dissertation, we provide algorithms to address both of these issues.

One approach to ameliorating selfishness in large networks is the idea of a mediator. A mediator implements a correlated equilibrium when it proposes a strategy to each player privately such that the mediator's proposal is the best interest for every player to follow. In this dissertation, we present a mediator that implements the best correlated equilibrium for an extended El Farol game. The extended El Farol game we consider has both positive and negative network effects. We study the degree to which this type of mediator can decrease the social cost. In particular, we give an exact characterization of *Mediation Value* (MV) and *Enforcement Value* (EV) for this game. MV measures the efficiency of our mediator compared to the best Nash equilibrium, and EV measures the efficiency of our mediator compared to the optimal social cost. This sort of exact characterization is uncommon for games with both kinds of network effects. An interesting outcome of our results is that both the MV and EV values can be unbounded for our game.

Recent years have seen significant interest in designing networks that are *self-healing* in the sense that they can automatically recover from adversarial attacks. Previous work shows that it is possible for a network to automatically recover, even when an adversary repeatedly deletes nodes in the network. However, there have not yet been any algorithms that self-heal in the case where an adversary takes over nodes in the network. In this dissertation, we address this gap. In particular, we describe a communication network over n nodes that ensures the following properties, even when an adversary controls up to $t \leq (1/4 - \epsilon)n$ nodes, for any constant $\epsilon > 0$. First, the network provides point-to-point communication with message cost and latency that are asymptotically optimal in an amortized sense. Second, the expected total number of message corruptions is $O(t(\log^* n)^2)$, after which the adversarially controlled nodes are effectively quarantined so that they cause no more corruptions.

In the problem of reliable multiparty computation (RMC), there are n parties, each with an individual input, and the parties want to jointly and reliably compute a function f over n inputs, assuming that it is not necessary to maintain the privacy of the inputs. The problem is complicated by the fact that an omniscient adversary controls a hidden fraction of the parties. We describe a self-healing algorithm for this problem. In particular, for a fixed function f, with n parties and m gates, we describe how to perform RMC repeatedly as the inputs to f change. Our algorithm maintains the following properties, even when an adversary controls up to $t \leq (\frac{1}{4} - \epsilon)n$ parties, for any constant $\epsilon > 0$. First, our algorithm performs each reliable computation with the following amortized resource costs: $O(m + n \log n)$ messages, $O(m + n \log n)$ computational operations, and $O(\ell)$ latency, where ℓ is the depth of the circuit that computes f. Second, the expected total number of corruptions is $O(t(\log^* n)^2)$.

Our empirical results show that the message cost reduces by up to a factor of 60 for communication and a factor of 65 for computation, compared to algorithms of no self-healing.

Contents

List of Figures

List of Figures

List of Tables

Chapter 1

Introduction

"The Future Has Arrived - It's Just Not Evenly Distributed Yet."
– William Gibson

Selfishness and malice have negative influences on the performance of distributed systems. Selfish players in a game can reduce social welfare; and malicious nodes can seriously disrupt the network. In this thesis, we study selfishness and malice in distributed systems, and we provide solutions and algorithms to address these issues.

1.1 Selfishness

In a game theoretic setting, players are selfish, where they try to maximize their own utility, without concern for others. Unfortunately, social welfare can decrease significantly when the players are selfish compared to when they are benevolent.

In game theory, the Nash equilibrium [61] is one of the basic solution concepts for non-cooperative games [18, 62]. Nash equilibrium is a situation, in which each player settles on a strategy, where no incentive to change this strategy unilaterally. Note that there is no communication among the players, and no player reveals his own strategy.

The Price of Anarchy [49] and the Price of Stability [4] are concepts in game theory to measure how the efficiency of a system degrades due to selfish behavior of the players. The Price of Anarchy is the ratio of the social cost of the worst Nash equilibrium to the optimal social cost of benevolent and omniscient dictator. However, the Price of Stability is the ratio of the social cost of the best Nash equilibrium to the optimal social cost of benevolent and omniscient dictator.

In many games with a high Price of Stability, one way to reduce social cost is to find a mediator with less social cost compared to the best Nash equilibrium. Informally, a mediator is a trusted party that gives recommendations to the players, or it can even act on behalf of the players. Note that each player has free will to accept or deny the mediator's behavior.

Several papers [20, 21, 24, 27, 34, 42] have been conducted to analyze equilibria and develop mediators for various games. They measure how efficient their equilibria or mediators are in terms of price of anarchy [49], price of stability [4], price of mediation [20] and value of correlation [8].

In this thesis, we study the El Farol game [5, 23, 26, 50] after we extend it to have both types of network effects [28]. We design a mediator that implements the best correlated equilibrium [10] of our game, and we measure how efficient our mediator is, in terms of the value of correlation.

The classical El Farol game is one of the non-cooperative games [18, 62], where no player cooperates with any other player and each player selects his action unilaterally. In this game, each player chooses either to go to the El Farol restaurant or stay home. If any player decides to stay, he has to pay a constant cost. If any player chooses to go, he has to pay a cost which increases linearly as the total number of players that go increases. This shows that the classical El Farol game has only a negative network effect [28].

However, many social and economic applications have both positive and negative network effects [28]. Note that the positive network effect is that the more players that do the same action, the less cost each one of them pays. In this thesis, we combine the positive and negative network effects in one game, where we let the utility function, of the “go” action be as follows. The more players that go, the less the cost of each player, up to a certain point at which the cost of each player starts to increase due to congestion. The utility function of the “stay” action remains a positive constant. In this way, we generalize the classical El Farol game to have both positive and negative network effects.

In this thesis, we employ a mediator for a game that has both positive and negative network effects, in order to improve the social welfare of this game, compared to the best Nash equilibrium. We consider the type of mediator that implements correlated equilibrium (CE) [10].

Correlated equilibrium is a probability distribution on a set of strategy profiles[1]. Note that this probability distribution is known to all players. One strategy profile is selected according to the probability distribution without announcing this selection to the players, and it is used to recommend each player with a strategy. In correlated equilibrium, each player has no incentive to deviate from the recommended strategy assuming that all other players follow the recommendations.

Our mediator is a trusted coordinator, which uses a correlated equilibrium as follows. First, this mediator selects one strategy profile according to the probability distribution of the correlated equilibrium. Second, it makes use of the selected strategy profile to recommend each player privately and separately with a strategy, where no player has incentive to deviate unilaterally from the recommendation.

In Chapter 2, we describe our optimal mediator that implements the best correlated equilibrium of our extended El Farol game. In particular, we show that the best correlated equilibrium has two strategy profiles. Also, we describe how much our optimal mediator improves the social welfare of our game, compared to the best Nash equilibrium and the

[1]A strategy profile is a vector of strategies, one strategy for each player.

optimal social welfare.

In our analysis, we measure the efficiency of the best correlated equilibrium by the value of correlation [8]. The value of correlation has two metrics: the mediation value and the enforcement value. The mediation value measures the efficiency of the best correlated equilibrium compared to the best Nash equilibrium; and the enforcement value measures the efficiency of the best correlated equilibrium compared to the optimal social cost.

We show that the mediation value and the enforcement value can be unbounded, in our extended El Farol game. Note that when the mediation value is unbounded, the best correlated equilibrium is infinitely efficient compared to the best Nash equilibrium. Further, when the enforcement value is unbounded, it shows that the best correlated equilibrium is really inefficient compared to the optimal social cost.

1.2 Malice

A fault is a malfunction or a deviation from expected behavior. Faults can occur due to a variety of reasons: hardware failure, power outage, software bugs, network problems and attacks. These faults may be either fail-stop failures (crashes) or Byzantine failures. In crashes, when a faulty node fails, it stops functioning and produces no output. Byzantine faults are due to the presence of an adversary that can take over a subset of nodes (up to some fraction of all nodes) and cause these nodes to behave in an arbitrary manner in order to disrupt the system.

We consider fault tolerance to deal with these faults. Fault tolerance is the property of enabling systems to function properly in the occurrence of failures. Byzantine fault tolerance is more challenging than fail-stop fault tolerance.

For the fail-stop model, we can use acknowledgements in synchronous networks to detect crashes without assuming a Public Key Infrastructure (PKI) [3, 80]. In particular, if a sender sends a message through a path of nodes to a receiver and the sender does not receive an acknowledgement from the receiver in a known roundtrip time, then it is considered due to a failure.

For Byzantine faults, the messages and acknowledgements can be signed (assuming a PKI) to detect message corruptions. However, how could we detect corruptions and recover networks without assuming a PKI in the presence of Byzantine nodes? In this thesis, we answer this question in the affirmative.

There are ways to implement fault tolerance in the presence of Byzantine faults. A simple approach is replication. In replication, several nodes redundantly perform the same tasks. If their outputs are different, then there is a corruption. Unfortunately, replication is expensive in terms of communication complexity and hardware cost. Another approach is *self-healing*. We say that a network is *self-healing* when it detects corruptions, inspects the corruption situation and recovers automatically from failures. Self-healing algorithms spend resources only when necessary.

Many self-healing algorithms [19, 37, 38, 64, 73, 75] have been proposed to enable networks to handle fail-stop faults. To the best of our knowledge, no self-healing algorithm has yet been designed to handle Byzantine faults. In this thesis, we provide self-healing algorithms to recover communication and computation networks in the presence of Byzantine faults.

In communication networks, a sender sends a message through a path of nodes to a receiver. In the presence of Byzantine nodes, the message can be dropped or corrupted if the path has at least one Byzantine node.

In Chapter 3, we develop a self-healing algorithm to recover communication networks from Byzantine faults. In this algorithm, we let the sender send the message through a path of nodes to the receiver, with some probability of triggering an algorithm that checks if there is a corruption occurred during the message transmission. If a corruption is detected, then a heavy-weight procedure is called. This procedure inspects the corruption situation to identify the nodes that corrupt the message; and so it quarantines these nodes. After all bad nodes are quarantined, no more corruption occurs in the presence of static adversary.

The following fact is used for corruption detection in communication networks: if a coin is flipped x times, where a coin is tail with constant probability, then the probability of having a substring of tails of length $\Omega(\log x)$ is at most $1/2$. By this fact, we show that it is expected that each interval of bad nodes, in which the adversary can corrupt the message, shrinks logarithmically from round to round. After $O(\log^* x)$ rounds [2], the interval of bad nodes shrinks to size zero, at which the corruption is detected.

Moreover, we show that each time a corruption is detected, there is a progress of quarantining Byzantine nodes. This implies that the number of times of calling the heavy-weight procedure is bounded in an amortized sense.

Using these ideas, we are able to achieve the following result with our self-healing algorithm. Assume we have a network with n nodes and $t \leq (1/4-\epsilon)n$ Byzantine nodes, for any constant $\epsilon > 0$. Then the network provides a point to point communication of path length ℓ, with expected number of messages $O(\ell + \log n)$ and expected latency $O(\ell)$ in an amortized sense. The expected total number of message corruptions is $O(t(\log^* n)^2)$, after which the Byzantine nodes are effectively quarantined so that they cause no more corruptions.

In computation networks, the computation is performed through a circuit of gates. Recall that in the presence of Byzantine attacks, the Byzantine adversary can take over a subset of gates (up to some fraction of the total number of gates) in order to disrupt the network.

In Chapter 4, we develop an algorithm that self-heals computation networks in the presence of Byzantine faults. The challenging part of this algorithm is to prove the following lemma in order to detect corruptions in circuits. Given a directed acyclic graph of n nodes, where each node survives with a constant probability, then the probability of having a rooted

[2]Note that $\log^* x$ is the iterated logarithm function, which is an extremely slowly growing function.

subgraph, of surviving nodes of size $\Omega(\log n)$, is at most $1/2$. We prove this lemma, and so we show that any rooted subgraph of bad nodes that produces incorrect output at the root, is expected to shrink logarithmically from round to round. This implies that after $O(\log^* n)$ rounds, any rooted subgraph of bad nodes, of size n, shrinks to size 0, at which the corruption is detected.

Recall that when any corruption is detected, a recovery procedure is triggered in order to quarantine the bad nodes. After all bad nodes are quarantined, the network is completely healed, i.e., no more corruption occurs.

We thus obtain the following result. Assume we have n parties providing inputs to a function f that can be computed by an arithmetic circuit with depth ℓ and containing m gates. Then our algorithm performs a reliable computation with $O(m + n \log n)$ messages sent by all parties, $O(m + n \log n)$ computational operations performed by all parties, and $O(\ell)$ latency. Moreover, the expected total number of times the computation returns a corrupted output is $O(t(\log^* n)^2)$.

1.3 Dissertation Organization

In this thesis, we describe solutions to some problems caused by selfishness and malice. In Chapter 2, we extend the El Farol game to have positive and negative network effects, and we provide a full characterization of the optimal mediator, which implements the best correlated equilibrium of this game. Chapter 3 describes Byzantine faults in communication networks, and we provide a self-healing algorithm to recover these networks. In Chapter 4, we develop a self-healing algorithm to recover computation networks from Byzantine faults. Finally, we conclude and discuss open problems in Chapter 5.

Chapter 2

Selfishness in Distributed Systems

"Power has only one duty - to secure the social welfare of the People."

– Benjamin Disraeli

2.1 Introduction

When players act selfishly to minimize their own costs, the outcome with respect to the social cost may be poor. The Price of Anarchy [49] and the Price of Stability [4] measure the impact of selfishness on the social cost. In a game with a high Price of Stability, one way to reduce social cost is to employ an appropriate mediator.

In the literature, there are several types of mediators [7, 27, 33, 34, 54, 66, 69, 70, 71, 79]. In this chapter, we consider the type of mediator that implements correlated equilibrium (CE) [10]. Our mediator is a trusted external coordinator that gives a recommendation to each player in such a way that no player has incentive to deviate from the recommendation.

We assume that the players are symmetric in the sense that they have the same utility function and the probability the mediator suggests a strategy to some player is independent of the identity of that player.

Ashlagi et al. [8] define two metrics to measure the efficiency of mediators: the mediation value (MV) and the enforcement value (EV). In this chapter, we compute these values, adapted for games where players seek to minimize the social cost. The *Mediation Value* is defined as the ratio of the minimum social cost over all Nash equilibria to the minimum social cost over all mediators. The *Enforcement Value* is the ratio of the minimum social cost over all mediators to the optimal social cost.

A mediator is optimal when its expected social cost is minimum over all mediators. Thus, the *Mediation Value* measures the efficiency of the optimal mediator with respect to the best Nash equilibrium; and the *Enforcement Value* measures the efficiency of the optimal mediator with respect to the optimal social cost.

In this chapter, we design an optimal mediator, which implements the best correlated equilibrium for a variant of the El Farol game. We measure the efficiency of our mediator in terms of the mediation value and the enforcement value. Also, we show that the mediation value and the enforcement value can be unbounded in our game.

2.1.1 El Farol Game

First we describe the classical El Farol game [5, 23, 26, 50]. El Farol is a tapas bar in Santa Fe. Every Friday night, a population of people decide whether or not to go to the bar. If too many people go, they will all have a worse time than if they stayed home, since the bar will be too crowded. That is a negative network effect [28].

Now we provide an extension of the classical El Farol game, where both negative and positive network effects [28] are considered. The positive network effect, we add to the game, is that if too few people go, those that go will also have a worse time than if they stayed home, since the bar will be too boring.

Motivation

Our motivation for studying this problem comes from the following discussion in [28].

"It's important to keep in mind, of course, that many real situations in fact display both kinds of [positive and negative] externalities - some level of participation by others is good, but too much is bad. For example, the El Farol Bar might be most enjoyable if a reasonable crowd shows up, provided it does not exceed 60. Similarly, an on-line social media site with limited infrastructure might be most enjoyable if it has a reasonably large audience, but not so large that connecting to the Web site becomes very slow due to the congestion."

We note that our extension of El Farol game is one of the simplest, non-trivial problems for which a mediator can improve the social cost. Thus, it is useful for studying the power of a mediation.

Formal Definition of the Extended El Farol Game

Now we formally define our game, which is non-atomic [11, 77], in the sense that no individual player has significant influence on the outcome. The number of players, n, is very large tending to infinity. The (c, s_1, s_2)-El Farol game has three parameters c, s_1 and s_2, where $0 < c < s_1$ and $s_2 > 0$. Note that c represents the individual cost for a player that goes when all other players stay; s_1 is the decline rate of the individual cost to go; s_2 is the growth rate of the individual cost to go; and c/s_1 represents the critical fraction of players that go, at which the rate of the individual cost to go changes. In particular, if x is the fraction of players to go, then the cost $f(x)$ for any player to go is as follows:

$$f(x) = \begin{cases} c - s_1 x & 0 \leq x \leq \frac{c}{s_1}, \\ s_2(x - \frac{c}{s_1}) & \frac{c}{s_1} \leq x \leq 1. \end{cases} \tag{2.1}$$

and the cost to stay is 1. The function $f(x)$ is illustrated in the two plots of Figure 2.1.

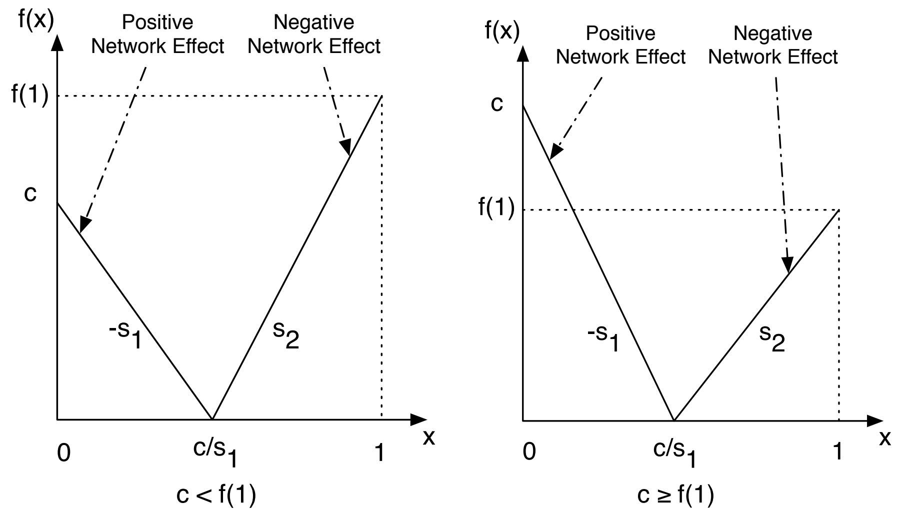

Figure 2.1: Cost to go, $f(x)$, for any individual player

Our Contributions

The main contributions of this chapter are threefold:

- We design an optimal mediator, which implements the best correlated equilibrium for an extension of the El Farol game with symmetric players. Notably, this extension incorporates both negative and positive network effects.
- We give an exact characterization of the *Mediation Value* (MV) and the *Enforcement Value* (EV) for our game.
- We show that both the MV and EV values can be unbounded for our game.

These main contributions were first presented as an extended abstract in [52].

2.1.2 Chapter Organization

In Section 2.2, we discuss the related work. Section 2.3 states the definitions and notations that we use in the El Farol game. Our results are given in Section 2.4, where we show our main theorem that characterizes the best correlated equilibrium, and we compute accordingly the *Mediation Value* and the *Enforcement Value*. The proof of Theorem 2.4.3 is shown in Section 2.5.

2.2 Related Work

Christodoulou and Koutsoupias [24] analyze the price of anarchy and the price of stability for Nash and correlated equilibria in linear congestion games. A consequence of their results is that the EV for these games is at least 1.577 and at most 1.6, and the MV is at most 1.015.

Brandt et al. [21] compute the mediation value and the enforcement value in ranking games. In ranking games, every outcome is a ranking of the players, and each player strictly prefers high ranks over lower ones [22]. They show that for the ranking games with $n > 2$ players, $EV = n - 1$. They also show that $MV = n - 1$ for $n > 3$ players; and for $n = 3$ players, at least one player has more than two actions.

The authors of [27] design a mediator that implements a correlated equilibrium for a virus inoculation game [9, 55]. In this game, there are n players, each corresponding to a node in a square grid. Every player has either to inoculate itself (at a cost of 1) or to do nothing and risk infection, which costs $L > 1$. After each node decides to inoculate or not, one node in the grid selected uniformly at random is infected with a virus. Any node, v, that chooses not to inoculate becomes infected if there is a path from the randomly selected node to v that traverses only uninoculated nodes. A consequence of their result is that EV is $\Theta(1)$ and MV is $\Theta((n/L)^{1/3})$ for this game.

Jiang et al. [42] analyze the price of miscoordination (PoM) and the price of sequential commitment (PoSC) in security games, which are defined to be a certain subclass of Stackelberg games. A consequence of their results is that MV is unbounded in general security games and it is at least 4/3 and at most $\frac{e}{e-1} \approx 1.582$ in a certain subclass of security games.

We note that a poorly designed mediator can make the social cost worse than what is obtained from the Nash equilibria. Bradonjic et al. [20] describe the *Price of Mediation* (PoM) which is the ratio of the social cost of the worst correlated equilibrium to the social cost of the worst Nash equilibrium. They show that for a simple game with two players and two possible strategies, PoM can be as large as 2. Also, they show for games with more players or more strategies per player that PoM can be unbounded.

Papadimitriou and Roughgarden [65] develop polynomial time algorithms for finding correlated equilibria in a broad class of succinctly representable multiplayer games. Unfortunately, their results do not extend to non-atomic games; moreover, they do not allow for direct computation of MV and EV, even when they can find the best correlated equilibrium.

Abraham et al. [1, 2] describe a distributed algorithm that enables a group of players to simulate a mediator. This algorithm works robustly with up to linear size coalitions, and up to a constant fraction of adversarial players. The result suggests that the concept of mediation can be useful even in the absence of a trusted external party.

In all equilibria above, the mediator does not act on behalf of the players. However, a more powerful type of mediators is described in [7, 33, 34, 54, 66, 69, 70, 71, 79], where a mediator can act on behalf of the players that give that right to it.

For multistage games, the notion of the correlated equilibrium is generalized to the communication equilibrium in [32, 58]. In communication equilibrium, the mediator implements a multistage correlated equilibrium, where it communicates with the players privately to receive their reports at every stage and accordingly selects the recommended strategy for each player.

In this chapter, we design a mediator that implements the best correlated equilibrium for a game that has both positive and negative network effects. Also, we measure how efficient our mediator is, in terms of the value of correlation [8].

2.3 Definitions and Notations

Now we state the definitions and notations that we use in the El Farol game.

Definition 2.3.1. A strategy profile *is a vector of strategies, one strategy for each player.*

Definition 2.3.2. *A configuration, $C(x)$, characterizes that a fraction of players, x, is being advised to go; and the remaining fraction of players, $(1-x)$, is being advised to stay.*

Note that any configuration has at least one strategy profile. For instance, for n players, $C(1/2)$ has $\binom{n}{n/2}$ different strategy profiles.

Definition 2.3.3. *A configuration distribution, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$, is a probability distribution over $k \geq 1$ configurations, where $(C(x_i), p_i)$ represents that configuration $C(x_i)$ is selected with probability p_i, for $1 \leq i \leq k$. Note that $0 \leq x_i \leq 1$, $0 < p_i \leq 1$, $\sum_{i=1}^{k} p_i = 1$ and if $x_i = x_j$ then $i = j$ for $1 \leq i, j \leq k$.*

For any player i, let $\mathcal{E}_G^i$ be the event that player i is advised to go, and C_G^i be the cost for player i to go (assuming that all other players conform to the advice). Also let $\mathcal{E}_S^i$ be the event that player i is advised to stay, and C_S^i be the cost for player i to stay. Since the players are symmetric, we will omit the index i.

A configuration distribution, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$, is a correlated equilibrium iff $\mathbf{E}[C_S|\mathcal{E}_G] \geq \mathbf{E}[C_G|\mathcal{E}_G]$ *and* $\mathbf{E}[C_G|\mathcal{E}_S] \geq \mathbf{E}[C_S|\mathcal{E}_S]$.

Definition 2.3.4. *A mediator is a trusted external party that uses a configuration distribution to advise the players such that this configuration distribution is a correlated equilibrium.*

The set of configurations and the probability distribution are known to all players. The mediator secretly selects a configuration according to the probability distribution, i.e., without announcing this selection to the players. The advice the mediator sends to a particular player, based on the selected configuration, is known only to that player.

2.4 Our Results

Our results are shown as follows. In Sections 2.4.1 and 2.4.2, we show the optimal social cost when all players are benevolent, and the minimum social cost over all Nash equilibria, for our extended El Farol game. In Section 2.4.3, we state our main theorem, Theorem 2.4.3, which characterizes the best correlated equilibrium and determines the *Mediation Value* and *Enforcement Value*. Also, we provide corollaries showing when the best correlated equilibrium is the best Nash equilibrium and when the *Mediation Value* and *Enforcement Value* are unbounded.

In Sections 2.4.1, 2.4.2 and 2.4.3, we assume that the *cost to stay* is 1. In Section 2.4.4, we justify this assumption. Moreover, in Section 2.4.5, we justify the assumption that both the line segment of negative slope and the line segment of positive slope intersect with the x-axis intersect at one point $(\frac{c}{s_1}, 0)$.

2.4.1 Optimal Social Cost

Lemma 2.4.1. *For any (c, s_1, s_2)-El Farol game, the optimal social cost is $(y^* f(y^*) + (1 - y^*))n$, where* $y^* = \begin{cases} \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) & \text{if } \frac{c}{s_1} \leq \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) \leq 1, \\ \frac{c}{s_1} & \text{if } \frac{c}{s_1} > \frac{1}{s_2}, \\ 1 & \text{otherwise.} \end{cases}$

Proof. By Equation (2.1), $f(x)$ has two cases. Let $f_1(x)$ be $f(x)$ for $x \in [0, \frac{c}{s_1}]$, and let

$f_2(x)$ be $f(x)$ for $x \in [\frac{c}{s_1}, 1]$. Also let $h_1(x)$ be the social cost when $0 \leq x \leq \frac{c}{s_1}$, and let $h_2(x)$ be the social cost when $\frac{c}{s_1} \leq x \leq 1$. Thus, $h_1(x) = (xf_1(x) + (1-x))n$ and $h_2(x) = (xf_2(x) + (1-x))n$.

We know that $h_1(x)$ is minimized at $x = \frac{c}{s_1}$. Further, we know that $h_2(x)$ is a quadratic function with respect to x, and thus it has one minimum over $x \in [\frac{c}{s_1}, 1]$ at $x = y^*$, where:

$$y^* = \begin{cases} \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) & \text{if } \frac{c}{s_1} \leq \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) \leq 1, \\ \frac{c}{s_1} & \text{if } \frac{c}{s_1} > \frac{1}{s_2}, \\ 1 & \textit{otherwise.} \end{cases}$$

Now let h^* be the optimal social cost. Thus, $h^* = min(h_1(\frac{c}{s_1}), h_2(y^*))$. Since $f_1(\frac{c}{s_1}) = f_2(\frac{c}{s_1})$, we have $h_1(\frac{c}{s_1}) = h_2(\frac{c}{s_1})$. Hence, we obtain $h^* = min(h_2(\frac{c}{s_1}), h_2(y^*))$. This implies that $h^* = h_2(y^*)$. □

2.4.2 Best Nash Equilibrium

Lemma 2.4.2. *For any* (c, s_1, s_2)*-El Farol game, if* $f(1) \geq 1$*, then the best Nash equilibria are: 1) at which the cost to go in expectation is equal to the cost to stay; and 2) at which all players would rather stay; otherwise, the best Nash equilibrium is at which all players would rather go. The social cost of the best Nash equilibrium is* $\min(n, f(1) \cdot n)$*.*

Proof. There are two cases for $f(1)$ to determine the best Nash equilibrium.
Case 1: $f(1) \geq 1$. Let N_y be a Nash equilibrium with the minimum social cost over all Nash equilibria and with a y-fraction of players that go in expectation. If $f(y) > 1$, then at least one player of the y-fraction of players would rather stay. Also if $f(y) < 1$, then at least one player of the $(1-y)$-fraction of players would rather go. Thus, we must have $f(y) = 1$ or all players would rather stay. Assume that each player has a mixed strategy, where player i goes with probability y_i. Recall that N_y has a y-fraction of players that go in expectation. Thus, $y = \frac{1}{n}\sum_{i=1}^{n} y_i$. Then the social cost is $\sum_{i=1}^{n}(y_i f(y) + (1-y_i))$, or equivalently, n.

Case 2: $f(1) < 1$. In this case, the best Nash equilibrium is at which all players would rather go, with a social cost of $f(1) \cdot n$.

Therefore, the social cost of the best Nash equilibrium is $min(n, f(1) \cdot n)$. $\square$

2.4.3 Optimal Mediator

Theorem 2.4.3. *For any (c, s_1, s_2)-El Farol game, if $c \leq 1$, then the best correlated equilibrium is the best Nash equilibrium; otherwise, the best correlated equilibrium is $\mathcal{D}\{(C(0), p), (C(x^*), 1-p)\}$, where $\lambda(c, s_1, s_2) = c(\frac{1}{s_1} + \frac{1}{s_2}) - \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}}$,*

$$x^* = \begin{cases} \lambda(c, s_1, s_2) & \text{if } \frac{c}{s_1} \leq \lambda(c, s_1, s_2) < 1, \\ \frac{c}{s_1} & \text{if } \lambda(c, s_1, s_2) < \frac{c}{s_1}, \\ 1 & \text{otherwise.} \end{cases}$$

and $p = \frac{(1-x^*)(1-f(x^*))}{(1-x^*)(1-f(x^*))+c-1}$. Moreover,

1) *the expected social cost is $(p + (1-p)(x^* f(x^*) + (1 - x^*)))n$,*

2) *the Mediation Value* (MV) *is $\frac{\min(f(1),1)}{p+(1-p)(x^*f(x^*)+(1-x^*))}$ and*

3) *the Enforcement Value* (EV) *is $\frac{p+(1-p)(x^*f(x^*)+(1-x^*))}{y^*f(y^*)+(1-y^*)}$, where*

$$y^* = \begin{cases} \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) & \text{if } \frac{c}{s_1} \leq \frac{1}{2}(\frac{c}{s_1} + \frac{1}{s_2}) \leq 1, \\ \frac{c}{s_1} & \text{if } \frac{c}{s_1} > \frac{1}{s_2}, \\ 1 & \text{otherwise.} \end{cases} .$$

The proof of this theorem is deferred to the next Section.

The following two corollaries show when our mediation performs the same as the best Nash equilibrium.

Corollary 2.4.4. *For any (c, s_1, s_2)-El Farol game, if $c \leq 1$, then* $\mathrm{MV} = 1$.

Proof. By Theorem 2.4.3, if $c \leq 1$, then the best correlated equilibrium is the best Nash equilibrium. In particular, if $f(1) < 1$, then all players would rather go; otherwise, all players stay, or a fraction of players go so that the cost to go in expectation is equal to the cost to stay. This implies that $MV = 1$. $\square$

Now we show that for $c > 1$, if $\lambda(c, s_1, s_2) \geq 1$, then the best correlated equilibrium is the best Nash equilibrium, where all players would rather go.

Corollary 2.4.5. *For any (c, s_1, s_2)-El Farol game, if $c > 1$ and $\lambda(c, s_1, s_2) \geq 1$, then* $\mathrm{MV} = 1$.

Proof. By Theorem 2.4.3, if $c > 1$ and $\lambda(c, s_1, s_2) \geq 1$, then $x^* \to 1$ and $p \to 0$. Now we prove that if $\lambda(c, s_1, s_2) \geq 1$, then the best correlated equilibrium is the best Nash equilibrium of the case $f(1) < 1$ in Lemma 2.4.2, i.e., if $\lambda(c, s_1, s_2) \geq 1$, then $f(1)$ must be less than 1. To do so, we prove that $\lambda(c, s_1, s_2) \geq 1 \Rightarrow f(1) < 1$. Assume by way of contradiction that

$$\lambda(c, s_1, s_2) \geq 1 \Rightarrow f(1) \geq 1. \tag{2.2}$$

Recall that $f(1) = s_2(1 - \frac{c}{s_1})$. By rearranging the right hand side of implication 2.2 after substituting $f(1)$ with $s_2(1 - \frac{c}{s_1})$, we obtain $\lambda(c, s_1, s_2) \geq 1 \Rightarrow \frac{c}{s_1} + \frac{1}{s_2} \leq 1$, or equivalently, $\lambda(c, s_1, s_2) \geq 1 \Rightarrow \frac{c}{s_1} + \frac{1}{s_2} \leq \lambda(c, s_1, s_2)$. Recall that $\lambda(c, s_1, s_2) = c(\frac{1}{s_1} + \frac{1}{s_2}) - \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}}$. Thus, we have

$$\begin{aligned}
\lambda(c, s_1, s_2) \geq 1 \qquad & \Rightarrow \frac{c}{s_1} + \frac{1}{s_2} \leq c(\frac{1}{s_1} + \frac{1}{s_2}) - \sqrt{\frac{c(\frac{1}{s_1} + \frac{1}{s_2})(c-1)}{s_2}} \\
& \Rightarrow c - \sqrt{s_2 \cdot c(\frac{1}{s_1} + \frac{1}{s_2})(c-1)} \geq 1 \\
& \Rightarrow c - 1 \geq s_2 c(\frac{1}{s_1} + \frac{1}{s_2}) \\
& \Rightarrow s_2 \cdot \frac{c}{s_1} \leq -1,
\end{aligned}$$

which contradicts since s_1, s_2 and c are all positive.

Hence, for $\lambda(c, s_1, s_2) \geq 1$, we have $x^* \to 1$, $p \to 0$ and $f(1) < 1$. By Theorem 2.4.3, $MV \to 1$. $\square$

Now we show that MV and EV can be unbounded in the following two corollaries. Corollary 2.4.6 shows that the mediation can be infinitely beneficial over the best Nash equilibrium, i.e., $MV \to \infty$. Corollary 2.4.7 shows that the social cost of the best correlated equilibrium can be infinitely higher than the optimal social cost of benevolent players, i.e., $EV \to \infty$. Even so, the best correlated equilibrium has half the social cost of the best Nash equilibrium, i.e., $MV = 2$.

Corollary 2.4.6. *For any $(2+\epsilon, \frac{2+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, as $\epsilon \to 0$,* MV $\to \infty$.

Proof. For any $(2+\epsilon, \frac{2+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, we have $f(1) = 1$. By Theorem 2.4.3, we obtain $x^* = 1-\epsilon$, $f(x^*) = 0$ and $p = \frac{\epsilon}{1+2\epsilon}$ for $\epsilon \leq \frac{1}{2}(\sqrt{3}-1)$. Thus we have

$$\lim_{\epsilon\to 0} MV = \lim_{\epsilon\to 0} \frac{\min(f(1), 1)}{\frac{\epsilon}{1+2\epsilon} + \epsilon(\frac{1+\epsilon}{1+2\epsilon})} = \infty.$$

$\square$

Corollary 2.4.7. *For any $(1+\epsilon, \frac{1+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, as $\epsilon \to 0$,* EV $\to \infty$.

Proof. For any $(1+\epsilon, \frac{1+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, by Theorem 2.4.3, we obtain $x^* = 1 + \epsilon^2 - \epsilon\sqrt{1+\epsilon^2}$ and $f(x^*) = 1 + \epsilon - \sqrt{1-\epsilon^2}$. Then we have

$$p = \frac{(1-(1+\epsilon^2-\epsilon\sqrt{1+\epsilon^2}))(1-(1+\epsilon-\sqrt{1-\epsilon^2}))}{(1-(1+\epsilon^2-\epsilon\sqrt{1+\epsilon^2}))(1-(1+\epsilon-\sqrt{1-\epsilon^2}))+\epsilon}.$$

Also we have $y^* = 1-\epsilon$ and $f(y^*) = 0$ for $\epsilon \leq \frac{1}{2}$. Thus we have

$$\lim_{\epsilon\to 0} EV = \lim_{\epsilon\to 0} \frac{p + (1-p)(x^* f(x^*) + (1-x^*))}{y^* f(y^*) + (1-y^*)} = \infty.$$

$\square$

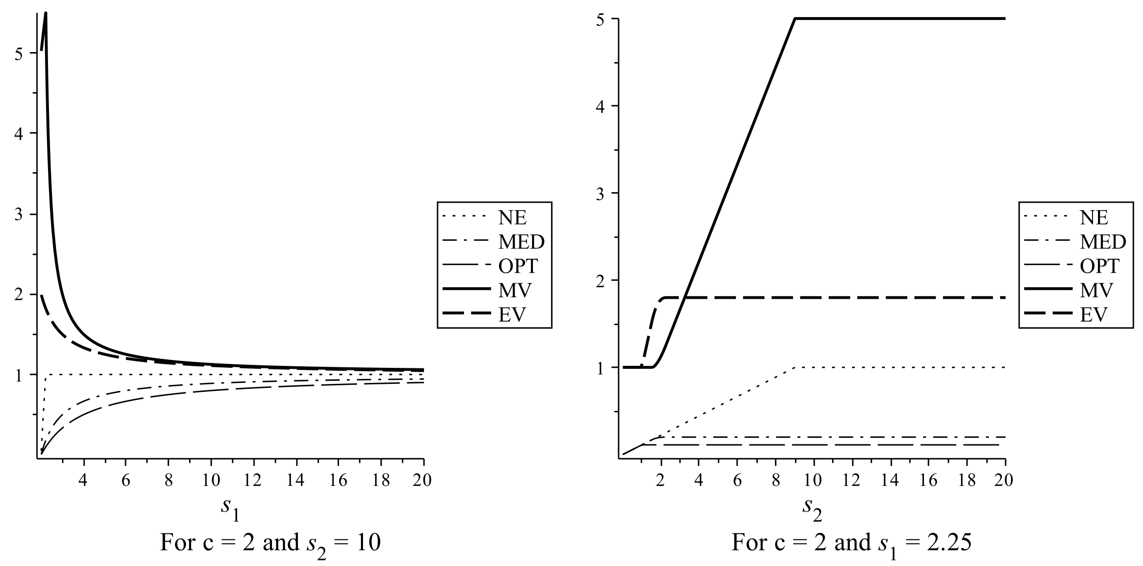

Figure 2.2: Mediation Metrics with respect to s_1 and s_2

Based on these results, we show in Figures 2.2 and 2.3: the social cost of the best Nash equilibrium (NE), the expected social cost of our optimal mediator (MED) and the optimal

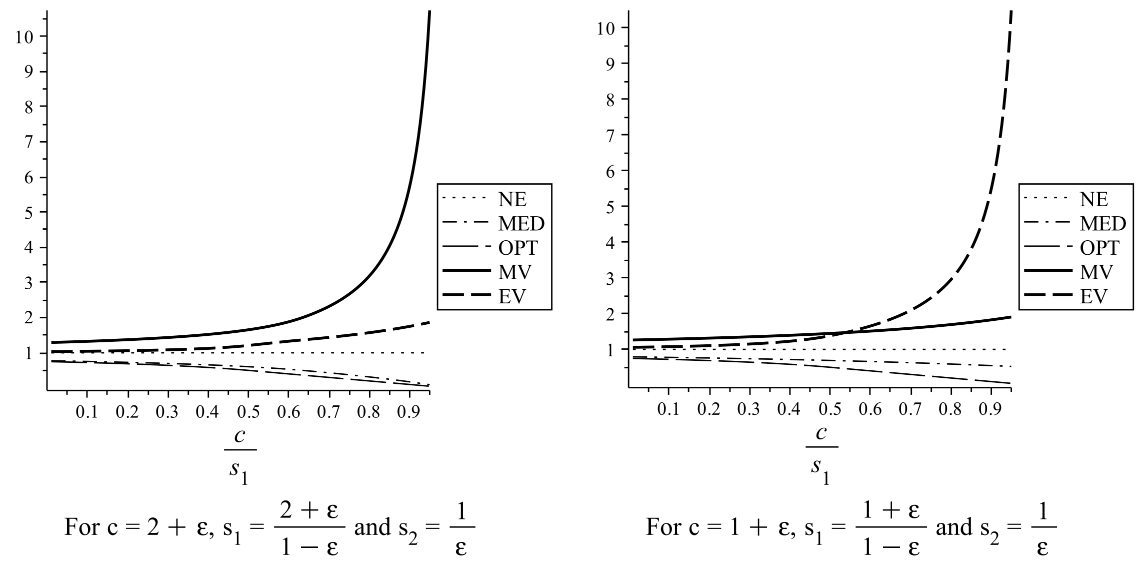

Figure 2.3: Mediation Metrics with respect to c/s_1

social cost (OPT), normalized by n, with respect to s_1, s_2 and c/s_1. Also we show the corresponding *Mediation Value* (*MV*) and *Enforcement Value* (*EV*).

In Figure 2.2, the left plot shows that for $c = 2$ and $s_2 = 10$, the values of NE, MED, OPT increase, each up to a certain point, when s_1 increases; however, the values of MV and EV decrease. Also, MV reaches its peak at the point where the best Nash equilibrium starts to remain constant with respect to s_1. In the right plot, we set $c = 2$ and $s_1 = 2.25$; it shows that the values of NE, MED, OPT, MV and EV increase, each up to a certain point, when s_2 increases.

Figure 2.3 illustrates Corollaries 2.4.6 and 2.4.7, and it shows how fast MV and EV go to infinity with respect to c/s_1, where $c/s_1 = 1 - \epsilon$. The left plot shows that for any $(2+\epsilon, \frac{2+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, as $c/s_1 \to 1$ (or $\epsilon \to 0$), $MV \to \infty$ and $EV \to 2$. In the right plot, for any $(1+\epsilon, \frac{1+\epsilon}{1-\epsilon}, \frac{1}{\epsilon})$-El Farol game, as $c/s_1 \to 1$ (or $\epsilon \to 0$), $EV \to \infty$ and $MV \to 2$.

Note that for any (c, s_1, s_2)-El Farol game, if $c/s_1 = 1$, then all players would rather go, with a social cost of 0. At this case, the best correlated equilibrium becomes the best Nash equilibrium. Hence, once c/s_1 reaches 1, MV suddenly drops to 1.

2.4.4 ***Cost to Stay*** Assumption

Now we justify our assumption that the cost to stay is unity. Let (c', s'_1, s'_2, t')-El Farol game be a variant of (c, s_1, s_2)-El Farol game, where $0 < c' < s'_1$, $s' > 0$ and the cost to stay is $t' > 0$. If x is the fraction of players to go, then the cost $f'(x)$ for any player to go is as follows:

$$f'(x) = \begin{cases} c' - s'_1 x & 0 \le x \le \frac{c'}{s'_1}, \\ s'_2(x - \frac{c'}{s'_1}) & \frac{c'}{s'_1} \le x \le 1. \end{cases}$$

The following lemma shows that any (c', s'_1, s'_2, t')-El Farol game can be reduced to a (c, s_1, s_2)-El Farol game.

Lemma 2.4.8. *Any (c', s'_1, s'_2, t')-El Farol game can be reduced to a (c, s_1, s_2)-El Farol game that has the same* Mediation Value *and* Enforcement Value*, where $c = \frac{c'}{t'}, s_1 = \frac{s'_1}{t'}$ and $s_2 = \frac{s'_2}{t'}$.*

Proof. In a manner similar to Theorem (2.4.3), for any (c', s'_1, s'_2, t')-El Farol game, if $c > t'$, then the best correlated equilibrium is $\mathcal{D}\{(C(0), p'), (C(x'), 1 - p')\}$, where

$$\lambda'(c', s'_1, s'_2, t') = c'(\frac{1}{s'_1} + \frac{1}{s'_2}) - \sqrt{\frac{c'(\frac{1}{s'_1} + \frac{1}{s'_2})(c' - t')}{s'_2}};$$

$$x' = \begin{cases} \lambda'(c', s'_1, s'_2, t') & \text{if } \frac{c'}{s'_1} \le \lambda'(c', s'_1, s'_2, t') < 1, \\ \frac{c'}{s'_1} & \text{if } \lambda'(c', s'_1, s'_2, t') < \frac{c'}{s'_1}, \\ 1 & \text{otherwise.} \end{cases}$$

and $p' = \frac{(1-x')(t'-f'(x'))}{(1-x')(t'-f'(x'))+c'-t'}$. Moreover,

1) the Mediation Value (MV') is $\frac{\min(f'(1),t')}{p't'+(1-p')(x'f'(x')+(1-x')t')}$ *and*

2) the Enforcement Value (EV') is $\frac{p't'+(1-p')(x'f'(x')+(1-x')t')}{y'f'(y')+(1-y')t'}$, where

$$y' = \begin{cases} \frac{1}{2}(\frac{c'}{s'_1} + \frac{t'}{s'_2}) & \text{if } \frac{c'}{s'_1} \le \frac{1}{2}(\frac{c'}{s'_1} + \frac{t'}{s'_2}) \le 1, \\ \frac{c'}{s'_1} & \text{if } \frac{c'}{s'_1} > \frac{t'}{s'_2}, \\ 1 & \textit{otherwise.} \end{cases}.$$

Similarly, for $c \le t'$, we have $MV' = 1$ and $EV' = \frac{\min(f'(1),t')}{y'f(y')+(1-y')t'}$.

For both cases, by Theorem 2.4.3, if we set $c = c'/t'$, $s_1 = s'_1/t'$ and $s_2 = s'_2/t'$, then we get $f'(x) = f(x) \cdot t'$, $y' = y^*$ and $\lambda'(c', s'_1, s'_2, t') = \lambda(c, s_1, s_2)$. This implies that $x' = x^*$ and $p' = p$. Thus, we obtain $MV' = MV$ and $EV' = EV$. □

2.4.5 Shifted-Up Cost

In the cost to go function, we assumed that both the line segment of negative slope and the line segment of positive slope intersect with the x-axis intersect at one point $(\frac{c}{s_1}, 0)$.

Now we analyze the case that the line segment of negative slope and the line segment of positive slope intersect at point $(\frac{c}{s_1}, a)$, where $a > 0$ is a constant. Note that the cost to stay is t_a. Now the cost to go is as follows.

$$f_a(x) = \begin{cases} c - s_1 x + a & 0 \leq x \leq \frac{c}{s_1}, \\ s_2(x - \frac{c}{s_1}) + a & \frac{c}{s_1} \leq x \leq 1. \end{cases}$$

We know that if $t_a \leq a$, then all players would rather stay. Now we analyze the case that $t_a > a$.

Compared to $(c, s_1, s_2, t_a - a)$-El Farol game in Section 2.4.4, we have the following. 1) The cost to stay increases by a; and 2) the individual cost to go increases by a for any fraction of players that go. Those imply that for any action and for any fraction of players that go, the individual cost increases by a positive constant a.

Therefore, this game of shifted-up cost can be reduced to $(c, s_1, s_2, t_a - a)$-El Farol game. In particular, after having the x, y and p of the best correlated equilibrium of $(c, s_1, s_2, t_a - a)$-El Farol game, we can obtain the following for the game of shifted-up cost. The best correlated equilibrium is $\mathcal{D}\{(C(0), p), (C(x), 1-p)\}$, with the following value of correlation:

$$MV = \frac{\min\left(f_a(1), t_a\right)}{pt_a + (1-p)(xf_a(x) + (1-x)t_a)}$$

and

$$EV = \frac{pt_a + (1-p)(xf_a(x) + (1-x)t_a)}{yf_a(y) + (1-y)t_a}.$$

2.5 Proof of Theorem 2.4.3

The proof of Theorem 2.4.3 has three main parts. First, we prove that if $c > 1$, then any optimal mediator has two configurations, and we give a full characterization for such an optimal mediator. Second, we prove that if $c \leq 1$, then the best correlated equilibrium is the best Nash equilibrium. Third, we characterize the *Mediation Value* and the *Enforcement Value.*

Recall that x_i is the fraction of players that are advised to go in configuration $C(x_i)$, which is selected with probability p_i in a configuration distribution, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$, for $1 \leq i \leq k$. We call a mediator over k configurations if the configuration distribution this mediator uses has k configurations. Also we define $\Delta(x)$ as the difference between the cost to stay and the cost to go at x, for any $0 \leq x \leq 1$. In particular, $\Delta(x) = 1 - f(x)$, where $f(x)$ is defined in Equation (2.1).

Now we state the notations that we use in the proof.

Definition 2.5.1. *BCE_i is a correlated equilibrium of i configurations with the least social cost over all correlated equilibria of i configurations.*

Definition 2.5.2. *$BCE_{\geq i}$ is a correlated equilibrium of $\geq i$ configurations with the least social cost over all correlated equilibria of $\geq i$ configurations.*

Definition 2.5.3. *BCE is the best correlated equilibrium over all correlated equilibria.*

2.5.1 Optimal Mediator for $c > 1$

In this section, we characterize an optimal mediator which implements BCE, for $c > 1$. In order to do so, we have three main parts.

First, we show that $BCE_{\geq 2} = BCE_2$. In particular, we show that $BCE_{\geq 2}$ has exactly two configurations: 1) one configuration that has a fraction of players advised to go such that this fraction lies on the positive network effect, i.e., the line segment of negative slope of $f(x)$ in Figure 2.1; and 2) one configuration that has a fraction of players advised to go such that this fraction lies on the negative network effect, i.e., the line segment of positive slope of $f(x)$ in Figure 2.1.

Second, we give a full characterization of $BCE_{\geq 2}$. In particular, we describe the probability distribution and the fraction of players advised to go for each configuration, in terms of c, s_1 and s_2.

Third, we show that $BCE_{\geq 2} \leq BCE_1$, i.e., $BCE_{\geq 2}$ has social cost of at most the social cost of the best Nash equilibrium. Also, we show that for some values of c, s_1 and s_2, $BCE_{\geq 2}$ reduces to BCE_1. In particular, the probability distribution of $BCE_{\geq 2}$ makes the optimal mediator select almost surely one configuration, where this configuration is one of the best Nash equilibria.

We start our proof by showing three properties of configuration distribution. These properties are used in the main parts of the proof.

Properties of Configuration Distribution

We first show when a configuration distribution is a correlated equilibrium.

Fact 2.5.4. *$\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium iff*

$$\sum_{i=1}^{k} p_i x_i \Delta(x_i) \geq 0 \tag{2.3}$$

and

$$\sum_{i=1}^{k} p_i (1 - x_i) \Delta(x_i) \leq 0. \tag{2.4}$$

Proof. Recall that $\mathcal{E}_G^i$ is the event that the mediator advises player i to go, C_G^i is the cost for player i to go, $\mathcal{E}_S^i$ is the event that the mediator advises player i to stay, and C_S^i is the cost for player i to stay. Also we will omit the index i since the players are symmetric.

By definition, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium iff

$$\mathbf{E}\left[C_S | \mathcal{E}_G\right] \geq \mathbf{E}\left[C_G | \mathcal{E}_G\right] \text{ and } \mathbf{E}\left[C_G | \mathcal{E}_S\right] \geq \mathbf{E}\left[C_S | \mathcal{E}_S\right].$$

Note that:

$$\mathbf{E}\left[C_S|\mathcal{E}_G\right] = 1,$$

$$\mathbf{E}\left[C_G|\mathcal{E}_G\right] = \frac{\sum_{i=1}^{k} p_i f(x_i) x_i}{\sum_{i=1}^{k} p_i x_i},$$

$$\mathbf{E}\left[C_G|\mathcal{E}_S\right] = \frac{\sum_{i=1}^{k} p_i f(x_i)(1-x_i)}{\sum_{i=1}^{k} p_i (1-x_i)},$$

and

$$E(C_S|\mathcal{E}_S) = 1.$$

Therefore, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium iff

$$\frac{\sum_{i=1}^{k} p_i f(x_i) x_i}{\sum_{i=1}^{k} p_i x_i} \leq 1 \tag{2.5}$$

and

$$\frac{\sum_{i=1}^{k} p_i f(x_i)(1-x_i)}{\sum_{i=1}^{k} p_i (1-x_i)} \geq 1. \tag{2.6}$$

By rearranging Inequalities (2.5) and (2.6), we have

$$\sum_{i=1}^{k} p_i x_i (1 - f(x_i)) \geq 0$$

and

$$\sum_{i=1}^{k} p_i (1-x_i)(1 - f(x_i)) \leq 0.$$

By definition, we know that $\Delta(x_i) = 1 - f(x_i)$. This implies the statement of the fact. □

In the following fact, we show the expected social cost of configuration distribution.

Fact 2.5.5. *The expected social cost of* $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$ *is*

$$(1 - \sum_{i=1}^{k} p_i x_i \Delta(x_i))n.$$

Proof. Let $Cost(C(x_i))$ be the cost of configuration $C(x_i)$ in $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$, for $1 \leq i \leq k$. We know that the expected social cost of $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$ is

$$\sum_{i=1}^{k} p_i Cost(C(x_i)).$$

We have $Cost(C(x_i)) = (x_i f(x_i) + (1 - x_i))n$, and since $\Delta(x_i) = 1 - f(x_i)$, it follows that $Cost(C(x_i)) = (1 - x_i \Delta(x_i))n$. Therefore, the expected social cost is

$$\sum_{i=1}^{k} p_i (1 - x_i \Delta(x_i))n,$$

or equivalently,

$$(\sum_{i=1}^{k} p_i - \sum_{i=1}^{k} p_i x_i \Delta(x_i))n.$$

Finally, we note that $\sum_{i=1}^{k} p_i = 1$. □

Recall that $\Delta(x)$ is the difference between the cost to stay and the cost to go when x players go. Now we show when $\Delta(x)$ of $C(x)$ is zero, positive and negative.

Fact 2.5.6. *For any configuration,* $C(x)$, *if* $c > 1$, *then*

$$\Delta(x) = 0 \Longleftrightarrow \left(x = \frac{c-1}{s_1}\right) \vee \left(x = \frac{1}{s_2} + \frac{c}{s_1} \wedge f(1) \geq 1\right);$$

$$\Delta(x) < 0 \Longleftrightarrow \left(0 \leq x < \frac{c-1}{s_1}\right) \vee \left(\frac{1}{s_2} + \frac{c}{s_1} < x \leq 1 \wedge f(1) > 1\right); \text{ and}$$

$$\Delta(x) > 0 \Longleftrightarrow \left(\frac{c-1}{s_1} < x < \frac{1}{s_2} + \frac{c}{s_1} \wedge f(1) \geq 1\right) \vee \left(\frac{c-1}{s_1} < x \leq 1 \wedge f(1) < 1\right).$$

Proof. Recall that $\Delta(x) = 1 - f(x)$. Then by Equation (2.1), we have

$$\Delta(x) = \begin{cases} \Delta_1(x) & 0 \leq x \leq \frac{c}{s_1}, \\ \Delta_2(x) & \frac{c}{s_1} \leq x \leq 1. \end{cases}$$

, where $\Delta_1(x) = 1 - (c - s_1 x)$ and $\Delta_2(x) = 1 - s_2(x - \frac{c}{s_1})$. Now we do a case analysis for x.

Case 1: if $0 \leq x \leq \frac{c}{s_1}$, then

$$\Delta_1(x) = 0 \iff x = \frac{c-1}{s_1};$$

$$\Delta_1(x) < 0 \iff 0 \leq x < \frac{c-1}{s_1};\ \textit{and}$$

$$\Delta_1(x) > 0 \iff \frac{c-1}{s_1} < x \leq \frac{c}{s_1}.$$

Case 2: if $\frac{c}{s_1} \leq x \leq 1$, then

$$\Delta_2(x) = 0 \iff x = \frac{1}{s_2} + \frac{c}{s_1} \wedge f(1) \geq 1;$$

$$\Delta_2(x) < 0 \iff \left(\frac{1}{s_2} + \frac{c}{s_1} < x \leq 1 \ \wedge \ f(1) > 1\right);\ \textit{and}$$

$$\Delta_2(x) > 0 \iff \left(\frac{c}{s_1} \leq x < \frac{1}{s_2} + \frac{c}{s_1} \wedge f(1) \geq 1\right) \vee \left(\frac{c}{s_1} \leq x \leq 1 \wedge f(1) < 1\right).$$

□

Now we describe the three main parts of the proof in detail.

(A) $BCE_{\geq 2} = BCE_2$

In Lemma 2.5.18, we show that $BCE_{\geq 2} = BCE_2$. Figure 2.4 shows the chart of facts and lemmas required to prove this lemma.

Now we show Lemma 2.5.11, where for any correlated equilibrium of $k \geq 2$ configurations with a configuration $C(x)$ where $x \in (0, \frac{c}{s_1})$, there exists a correlated equilibrium, with less

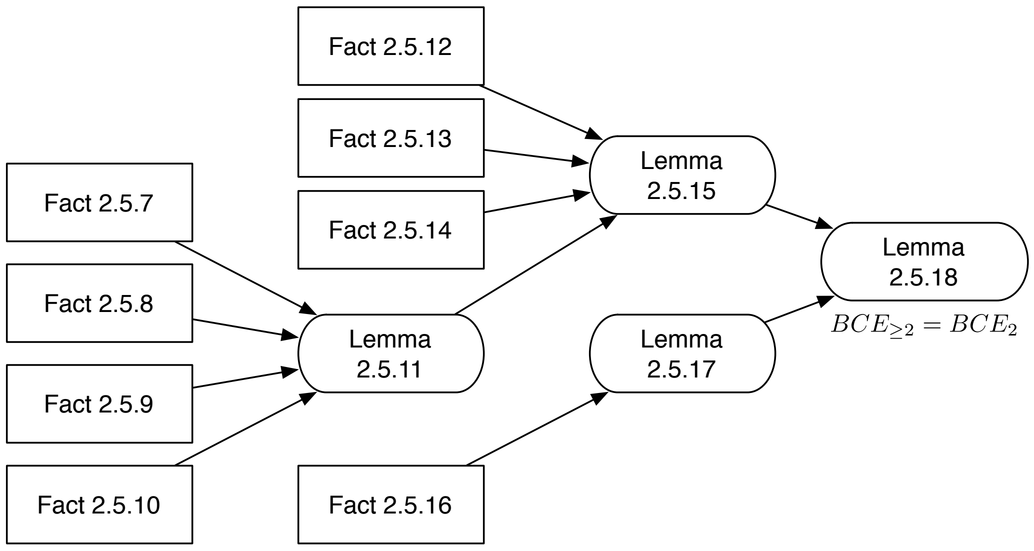

Figure 2.4: Proof Chart of Lemma 2.5.18

social cost, of k configurations with no configuration $C(x')$ where $x' \in (0, \frac{c}{s_1})$. The following four facts are used to prove this lemma. See Figure 2.5.

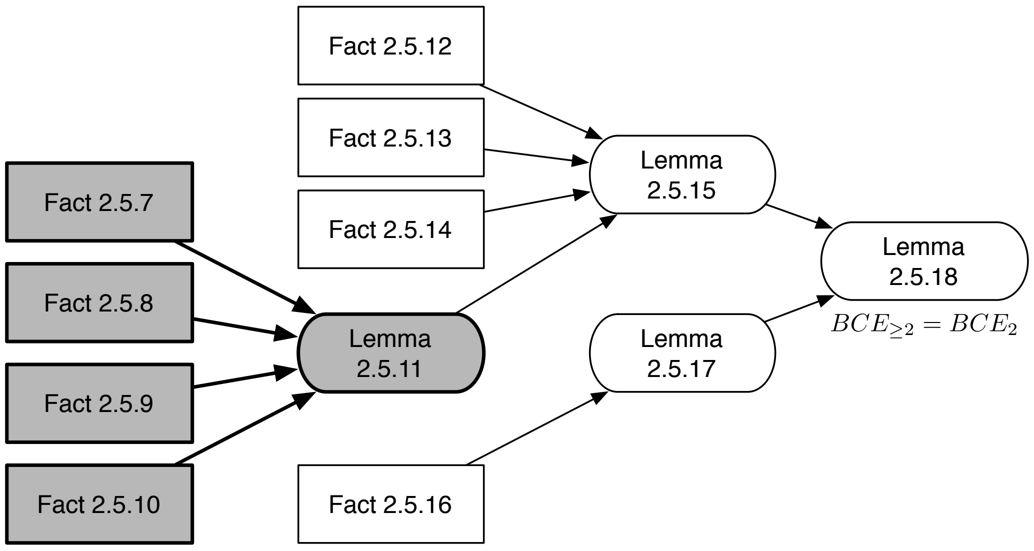

Figure 2.5: Proof Chart of Lemma 2.5.18 after proving Lemma 2.5.11

Fact 2.5.7. *For any correlated equilibrium,* $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, *where* $k \geq 2$ *and* $0 < x_j < \frac{c-1}{s_1}$*, there exists a correlated equilibrium,* $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$*, with less social cost, where* $x'_j = 0$*.*

Proof. Let M_k be a mediator that uses $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $0 < x_j < \frac{c-1}{s_1}$. By Fact 2.5.4, we have

$$p_j x_j \Delta(x_j) + \sum_{1 \leq i \leq k, i \neq j} p_i x_i \Delta(x_i) \geq 0 \tag{2.7}$$

and

$$p_j(1 - x_j)\Delta(x_j) + \sum_{1 \leq i \leq k, i \neq j} p_i(1 - x_i)\Delta(x_i) \leq 0. \tag{2.8}$$

Since $0 < x_j < \frac{c-1}{s_1}$, by Fact 2.5.6, $\Delta(x_j) < 0$. Now let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution that has $x'_j = 0$. Thus, we have $p_j x'_j \Delta(x'_j) = 0$ and $p_j x_j \Delta(x_j) < 0$. By Inequality (2.7), we have

$$p_j x'_j \Delta(x'_j) + \sum_{1 \leq i \leq k, i \neq j} p_i x_i \Delta(x_i) > 0. \tag{2.9}$$

We know that $\Delta(x'_j) < \Delta(x_j) < 0$ and $(1 - x'_j) > (1 - x_j) > 0$. Thus, we have

$$(1 - x'_j)\Delta(x'_j) < (1 - x_j)\Delta(x_j),$$

or equivalently,

$$\Delta(x'_j) < (1 - x_j)\Delta(x_j).$$

By Inequality (2.8), we get

$$p_j \Delta(x'_j) + \sum_{1 \leq i \leq k, i \neq j} p_i(1 - x_i)\Delta(x_i) < 0. \tag{2.10}$$

Now by Fact 2.5.4 and Inequalities (2.9) and (2.10), $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium. Let M'_k be a mediator that uses this correlated equilibrium. By Fact 2.5.5, and since $x'_j = 0$, the expected social cost of M'_k is

$$(1 - \sum_{1 \leq i \leq k, i \neq j} p_i x_i \Delta(x_i))n.$$

Moreover, by Fact 2.5.5, the expected social cost of M_k is

$$((1 - \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i)) - p_j x_j \Delta(x_j))n.$$

Since $\Delta(x_j) < 0$ and $x_j > 0$, the expected social cost of M'_k is less than the expected social cost of M_k. □

Fact 2.5.8. *If $f(1) \ge 1$, then for any correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $k \ge 2$ and $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$, there exists a correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$, with less social cost, where $\frac{c}{s_1} < x'_j < \frac{c}{s_1} + \frac{1}{s_2}$ and $f(x'_j) = f(x_j)$.*

Proof. Let M_k be a mediator that uses $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$. By Fact 2.5.4, we have

$$p_j x_j \Delta(x_j) + \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i) \ge 0 \tag{2.11}$$

and

$$p_j (1 - x_j) \Delta(x_j) + \sum_{1 \le i \le k, i \ne j} p_i (1 - x_i) \Delta(x_i) \le 0. \tag{2.12}$$

Recall that $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$ and $f(1) \ge 1$. Then $\exists x'_j : \frac{c}{s_1} < x'_j < \frac{c}{s_1} + \frac{1}{s_2}$ and $f(x'_j) = f(x_j)$. Now let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution. Since $f(x'_j) = f(x_j)$, $\Delta(x'_j) = \Delta(x_j)$. We know that $x'_j > x_j$, then $x'_j \Delta(x'_j) > x_j \Delta(x_i)$. By Inequality (2.11), we obtain

$$p_j x'_j \Delta(x'_j) + \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i) > 0.$$

Since $(1 - x'_j) < (1 - x_j)$, we have $(1 - x'_j)\Delta(x'_j) < (1 - x_j)\Delta(x_j)$. By Inequality (2.12), we get

$$p_j (1 - x'_j) \Delta(x'_j) + \sum_{1 \le i \le k, i \ne j} p_i (1 - x_i) \Delta(x_i) < 0.$$

Now by Fact 2.5.4, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium. Let M'_k be a mediator that uses this correlated equilibrium. By Fact 2.5.5, the expected social cost of M'_k is

$$((1 - \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i)) - p_j x'_j \Delta(x'_j))n,$$

and the expected social cost of M_k is

$$((1 - \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i)) - p_j x_j \Delta(x_j))n.$$

Since $p_j x'_j \Delta(x'_j) > p_j x_j \Delta(x_i)$, the expected social cost of M'_k is less than the expected social cost of M_k. □

Fact 2.5.9. *If $f(1) < 1$, then for any correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $k \ge 2$, $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$ and $f(x_j) \le f(1)$, there exists a correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$, with less social cost, where $\frac{c}{s_1} < x'_j \le 1$ and $f(x'_j) = f(x_j)$.*

Proof. We know that for $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$ and $f(x_j) \le f(1) < 1$, $\exists x'_j : \frac{c}{s_1} < x'_j \le 1$ and $f(x'_j) = f(x_j)$. In a manner similar to the proof of Lemma 2.5.8, we prove this Lemma. □

Fact 2.5.10. *If $f(1) < 1$, then for any correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $k \ge 2$, $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$ and $f(x_j) > f(1)$, there exists a correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$, with less social cost, where $x'_j = 1$.*

Proof. Let M_k be a mediator that uses $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $\frac{c-1}{s_1} < x_j < \frac{c}{s_1}$. By Fact 2.5.4, we have

$$p_j x_j \Delta(x_j) + \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i) \ge 0 \tag{2.13}$$

and

$$p_j(1 - x_j)\Delta(x_j) + \sum_{1\leq i\leq k, i\neq j} p_i(1 - x_i)\Delta(x_i) \leq 0. \tag{2.14}$$

Now let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution, where $x'_j = 1$. For $f(x_j) > f(x'_j)$ and $f(x'_j) < 1$, $\Delta(x'_j) > \Delta(x_j)$. We know that $x'_j > x_j$. Thus, $x'_j\Delta(x'_j) > x_j\Delta(x_i)$. By Inequality (2.13), we obtain

$$p_j x'_j \Delta(x'_j) + \sum_{1\leq i\leq k, i\neq j} p_i x_i \Delta(x_i) > 0.$$

Also, we know that $(1 - x_j) > 0$ and $\Delta(x_j) > 0$. Thus, $(1 - x_j)\Delta(x_j) > 0$. Since $(1 - x'_j) = 0$, we have

$$(1 - x'_j)\Delta(x'_j) < (1 - x_j)\Delta(x_j).$$

By Inequality (2.14), we get

$$p_j(1 - x'_j)\Delta(x'_j) + \sum_{1\leq i\leq k, i\neq j} p_i(1 - x_i)\Delta(x_i) < 0.$$

Now by Fact 2.5.4, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium. Let M'_k be a mediator that uses this correlated equilibrium. By Fact 2.5.5, the expected social cost of M'_k is

$$((1 - \sum_{1\leq i\leq k, i\neq j} p_i x_i \Delta(x_i)) - p_j x'_j \Delta(x'_j))n,$$

and the expected social cost of M_k is

$$((1 - \sum_{1\leq i\leq k, i\neq j} p_i x_i \Delta(x_i)) - p_j x_j \Delta(x_j))n.$$

Since $p_j x'_j \Delta(x'_j) > p_j x_j \Delta(x_i)$, the expected social cost of M'_k is less than the expected social cost of M_k. $\square$

Lemma 2.5.11. *For any correlated equilibrium of $k \geq 2$ configurations with a configuration $C(x)$ where $x \in (0, \frac{c}{s_1})$, there exists a correlated equilibrium, with less social cost, of k configurations with no configuration $C(x')$ where $x' \in (0, \frac{c}{s_1})$.*

Proof. Facts 2.5.7, 2.5.8, 2.5.9 and 2.5.10 prove the statement of this lemma. □

Now we show Lemma 2.5.15, where $BCE_{\geq 2}$ has one configuration of a fraction of players advised to go that lies on the positive network effect, and any other configuration has a fraction of players advised to go that lies on the negative network effect. To show Lemma 2.5.15, we first prove Facts 2.5.12, 2.5.13 and 2.5.14, as illustrated in Figure 2.6.

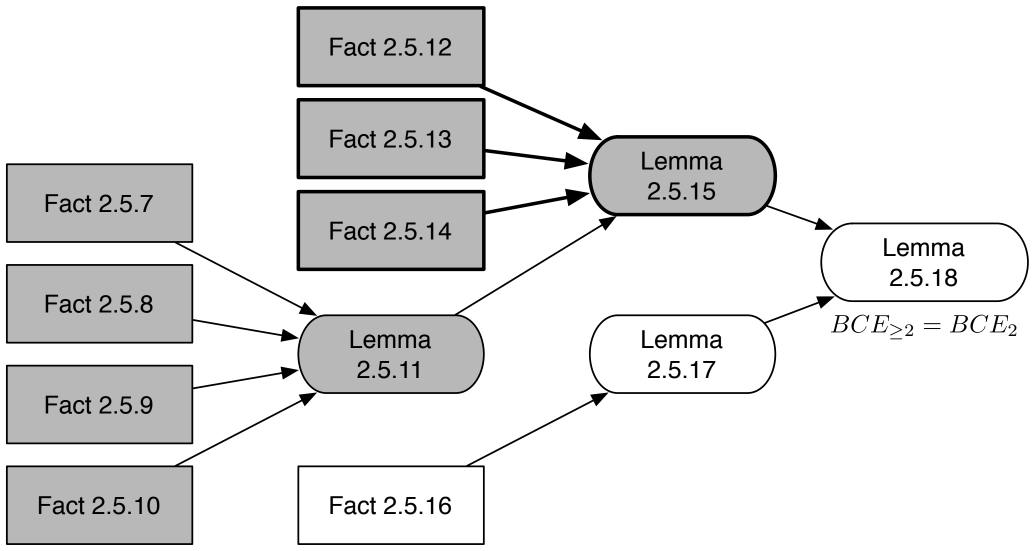

Figure 2.6: Proof Chart of Lemma 2.5.18 after proving Lemma 2.5.15

Fact 2.5.12 shows that for any correlated equilibrium of $k \geq 2$ configurations with a configuration $C(x)$ of $\Delta(x) = 0$, there exists a correlated equilibrium of $k-1$ configurations, that has less social cost, with no configuration $C(x')$ of $\Delta(x') = 0$. Facts 2.5.13 and 2.5.14 describe two properties of any correlated equilibrium that has at least two configurations, with no configuration $C(x')$ of $\Delta(x') = 0$.

Fact 2.5.12. *For any correlated equilibrium,* $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$ *, where* $k \geq 2$ *and* $\Delta(x_j) = 0$*, there exists a correlated equilibrium,* $\mathcal{D}\{(C(x_1), \frac{p_1}{1-p_j}), \ldots, (C(x_{j-1}), \frac{p_{j-1}}{1-p_j}), (C(x_{j+1}), \frac{p_{j+1}}{1-p_j}), \ldots, (C(x_k), \frac{p_k}{1-p_j})\}$*, with less expected social cost.*

Proof. Let M_k be a mediator that uses $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$ for $k \geq 2$, and there is some $1 \leq j \leq k$ such that $\Delta(x_j) = 0$.

We know that $k \geq 2$. Thus, $0 < p_j < 1$. Now let $\mathcal{D}\{(C(x_1), \frac{p_1}{1-p_j}), \ldots, (C(x_{j-1}), \frac{p_{j-1}}{1-p_j}), (C(x_{j+1}), \frac{p_{j+1}}{1-p_j}), \ldots, (C(x_k), \frac{p_k}{1-p_j})\}$ be a configuration distribution over $k-1$ configurations.

Since M_k is a mediator and $\Delta(x_j) = 0$, Constraints (2.3) and (2.4) of Fact 2.5.4 imply that

$$\sum_{1 \leq i \leq k, i \neq j} p_i x_i \Delta(x_i) \geq 0$$

and

$$\sum_{1 \leq i \leq k, i \neq j} p_i (1 - x_i) \Delta(x_i) \leq 0.$$

Now if we multiply both sides of these two constraints by $\frac{1}{1-p_j}$, we have

$$\sum_{1 \leq i \leq k, i \neq j} \frac{p_i}{1-p_j} x_i \Delta(x_i) \geq 0$$

and

$$\sum_{1 \leq i \leq k, i \neq j} \frac{p_i}{1-p_j} (1 - x_i) \Delta(x_i) \leq 0.$$

By Fact 2.5.4, $\mathcal{D}\{(C(x_1), \frac{p_1}{1-p_j}), \ldots, (C(x_{j-1}), \frac{p_{j-1}}{1-p_j}), (C(x_{j+1}), \frac{p_{j+1}}{1-p_j}), \ldots, (C(x_k), \frac{p_k}{1-p_j})\}$ is a correlated equilibrium. Let M_{k-1} be a mediator that uses this correlated equilibrium. By Fact 2.5.5, the expected social cost of M_{k-1} is

$$(1 - \frac{1}{1-p_j} \sum_{1 \leq i \leq k, i \neq j} p_i x_i \Delta(x_i))n,$$

and since $\Delta(x_j) = 0$, the expected social cost of M_k is

$$(1 - \sum_{1 \le i \le k, i \ne j} p_i x_i \Delta(x_i))n.$$

We know that $0 < p_j < 1 \implies \frac{1}{1-p_j} > 1$. Therefore, the expected social cost M_{k-1} is less than the expected social cost of M_k. □

Fact 2.5.13. *Any correlated equilibrium, of $k \ge 2$ configurations, with no configuration, $C(x)$ of $\Delta(x) = 0$, has at least one configuration $C(x_u)$ of $\Delta(x_u) > 0$ and at least one configuration $C(x_v)$ of $\Delta(x_v) < 0$, for $1 \le u, v \le k$.*

Proof. Assume that there exists a correlated equilibrium, of $k \ge 2$ configurations, with no configuration, $C(x)$ of $\Delta(x) = 0$.

Note that $0 < p_i < 1$ and $0 \le x_i \le 1$ for all $1 \le i \le k$. By Constraint (2.3) of Fact 2.5.4, there exists u such that $\Delta(x_u) > 0$ for $1 \le u \le k$. Also, Constraint (2.4) of Fact 2.5.4 implies that there exists v such that $\Delta(x_v) < 0$ for $1 \le v \le k$. □

Fact 2.5.14. *Any correlated equilibrium, of $k \ge 2$ configurations, with no configuration, $C(x)$ where $\Delta(x) = 0$, has $\sum_{1 \le i \le k, i \ne j} p_i \Delta(x_i)(x_i - x_j) \ge 0$, for all $1 \le j \le k$.*

Proof. Let M_k be a mediator that uses $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, with no configuration $C(x)$ of $\Delta(x) = 0$. Now fix any $1 \le j \le k$, and do a case analysis for $\Delta(x_j) \ne 0$.

Case 1: if $\Delta(x_j) < 0$, then by repeated application of Fact 2.5.4 we have

$$\frac{\sum_{1 \le i \le k, i \ne j} p_i \Delta(x_i)(1 - x_i)}{(1 - x_j)|\Delta(x_j)|} \le p_j \le \frac{\sum_{1 \le i \le k, i \ne j} p_i \Delta(x_i) x_i}{x_j |\Delta(x_j)|} \tag{2.15}$$

Removing p_j from Inequality (2.15) and rearranging, we get

$$x_j |\Delta(x_j)| \sum_{1 \le i \le k, i \ne j} p_i \Delta(x_i)(1 - x_i) \le (1 - x_j)|\Delta(x_j)| \sum_{1 \le i \le k, i \ne j} p_i \Delta(x_i) x_i.$$

By canceling the common terms, we have

$$\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_j \le \sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_i.$$

Case 2: if $\Delta(x_j) > 0$, then similarly by repeated application of Fact 2.5.4 we have

$$\frac{-\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_i}{x_j\Delta(x_j)} \le p_j \le \frac{-\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)(1-x_i)}{(1-x_j)\Delta(x_j)} \tag{2.16}$$

Removing p_j from Inequality (2.16) and rearranging, we get

$$x_j\Delta(x_j)\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)(1-x_i) \le (1-x_j)\Delta(x_j)\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_i.$$

By canceling the common terms, we have

$$\sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_j \le \sum_{1\le i\le k, i\ne j} p_i\Delta(x_i)x_i.$$

Since j is any value between 1 and k, this implies the statement of the lemma for every such j. □

Lemma 2.5.15. *For any (c, s_1, s_2)-El Farol game, $BCE_{\ge 2}$ has one configuration that has no players advised to go, and any other configuration has at least a $\frac{c}{s_1}$-fraction of players advised to go.*

Proof. Let $BCE_{\ge 2} = \mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_k), p_k)\}$. Now we prove that $BCE_{\ge 2}$ must have a configuration that has no player advised to go. We know by Facts 2.5.12 and 2.5.13 that $\exists j : \Delta(x_j) < 0$, where $1 \le j \le k$. By Fact 2.5.6, $\Delta(x_j) < 0$ iff ($x_j \in (1/s_2 + c/s_1, 1]$ *and* $f(1) > 1$) or $x_j \in [0, \frac{c-1}{s_1})$. Now we do a case analysis for x_j.

Case 1: $x_j \in (1/s_2 + c/s_1, 1]$ *and* $f(1) > 1$. Assume by way of contradiction that $BCE_{\ge 2}$ has no configuration that has less than a $\frac{c-1}{s_1}$-fraction of players advised to go. Let x_q be the smallest fraction that is $x_q > 1/s_2 + c/s_1$, where $1 \le q \le k$. By Facts 2.5.6 and 2.5.12, for $1 \le r \le k$,

$$\Delta(x_r) = \begin{cases} > 0 & \text{if } x_r < x_q, \\ < 0 & \text{otherwise.} \end{cases}$$

Note that by the definition of the configuration distribution, if $x_r = x_q$ then $r = q$. Therefore, we have

$$\sum_{1 \le r \le k, r \ne q} p_r \Delta(x_r)(x_r - x_q) < 0. \tag{2.17}$$

By Facts 2.5.12 and 2.5.14, Inequality (2.17) contradicts that $BCE_{\ge 2}$ has the least social cost over all correlated equilibria of $k \ge 2$ configurations. Thus, $BCE_{\ge 2}$ must have a configuration, $C(x)$, where $x < \frac{c-1}{s_1}$, and the rest of the argument is as in Case 2.

Case 2: $x_j \in [0, \frac{c-1}{s_1})$. By Fact 2.5.7, $x_j = 0$.

By the definition of the configuration distribution, $BCE_{\ge 2}$ has no two configurations that have the same fraction of players that are advised to go. Thus, $BCE_{\ge 2}$ has one configuration that has no players advised to go.

We know that $\Delta(\frac{c-1}{s_1}) = 0$. By Fact 2.5.12, $BCE_{\ge 2}$ has no configuration $C(\frac{c-1}{s_1})$. By Lemma 2.5.11, $BCE_{\ge 2}$ has no configuration with an x-fraction of players advised to go, where $x \in (0, \frac{c}{s_1})$. □

Now we show Lemma 2.5.17, where $BCE_{\ge 2}$ has at most one configuration lies on the negative network effect, and any other configuration lies on the positive network effect. In order to do so, as shown in Figure 2.7, we prove Fact 2.5.16. This fact shows a property for any configuration distribution which has at least two configurations that have fractions of players advised to go such that these fractions lie on the negative network effect.

Fact 2.5.16. *For any $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_i), p_i), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, and for any arbitrary x_i and x_j such that $x_j > x_i \ge \frac{c}{s_1}$, there exists $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_{i-1}), p_{i-1}), (C(x_{i+1}), p_{i+1}), \ldots, (C(x'_j), p_i + p_j), \ldots, (C(x_k), p_k)\}$, where $x'_j = \frac{p_i}{p_i+p_j} x_i + \frac{p_j}{p_i+p_j} x_j$. Moreover,*

1) $(p_i + p_j)x'_j \Delta(x'_j) > p_i x_i \Delta(x_i) + p_j x_j \Delta(x_j);$ *and*

2) $(p_i + p_j)(1 - x'_j)\Delta(x'_j) < p_i(1 - x_i)\Delta(x_i) + p_j(1 - x_j)\Delta(x_j).$

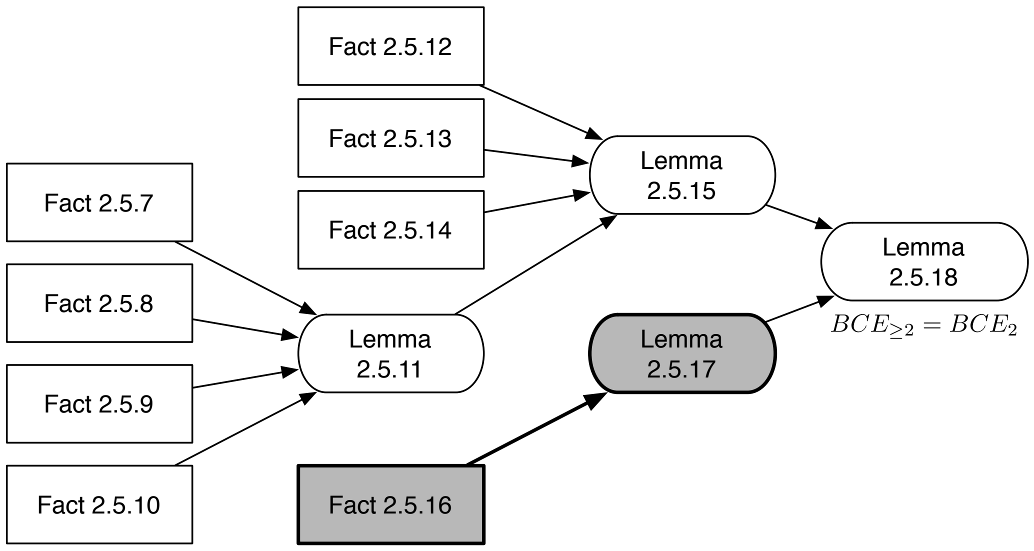

Figure 2.7: Proof Chart of Lemma 2.5.18 after proving Lemma 2.5.17

Proof. Let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_i), p_i), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution that has $x_j > x_i \geq \frac{c}{s_1}$. Also let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_{i-1}), p_{i-1}), (C(x_{i+1}), p_{i+1}) , \ldots, (C(x'_j), p_i + p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution that has $x'_j = \frac{p_i}{p_i+p_j}x_i + \frac{p_j}{p_i+p_j}x_j$. We know that $0 < p_i, p_j < 1$ and $x_j > x_i$. Thus $x_i < x'_j < x_j$. Assume by way of contradiction that

$$(p_i + p_j)x'_j\Delta(x'_j) \leq p_i x_i \Delta(x_i) + p_j x_j \Delta(x_j),$$

or equivalently,

$$x'_j\Delta(x'_j) \leq \frac{p_i}{p_i + p_j}x_i\Delta(x_i) + \frac{p_j}{p_i + p_j}x_j\Delta(x_j).$$

Let $p = \frac{p_i}{p_i+p_j}$, so $1 - p = \frac{p_j}{p_i+p_j}$. Then we have

$$x'_j\Delta(x'_j) \leq p x_i \Delta(x_i) + (1-p)x_j\Delta(x_j).$$

Recall that for $\frac{c}{s_1} \leq x \leq 1$, $\Delta(x) = 1 - s_2(x - \frac{c}{s_1})$. Since $\frac{c}{s_1} \leq x_i, x_j, x'_j \leq 1$, we get

$$x'_j(1 - s_2(x'_j - \frac{c}{s_1})) \leq p x_i(1 - s_2(x_i - \frac{c}{s_1})) + (1-p)x_j(1 - s_2(x_j - \frac{c}{s_1})).$$

Since $x'_j = px_i + (1-p)x_j$, we have

$$x'_j(-s_2(x'_j - \frac{c}{s_1})) \leq px_i(-s_2(x_i - \frac{c}{s_1})) + (1-p)x_j(-s_2(x_j - \frac{c}{s_1})).$$

We know that $s_2 > 0$, and hence dividing by $-s_2$, we get

$$x'_j(x'_j - \frac{c}{s_1}) \geq px_i(x_i - \frac{c}{s_1}) + (1-p)x_j(x_j - \frac{c}{s_1}).$$

Since $-\frac{c}{s_1}x'_j = -\frac{c}{s_1}(px_i + (1-p)x_j)$, we have

$$x'^2_j \geq px_i^2 + (1-p)x_j^2.$$

Substituting x'_j by $px_i + (1-p)x_j$, we get

$$p^2x_i^2 + 2p(1-p)x_ix_j + (1-p)^2x_j^2 \geq px_i^2 + (1-p)x_j^2.$$

By rearranging, we have

$$p(1-p)(x_j^2 - 2x_ix_j + x_i^2) \leq 0.$$

Now since $0 < p < 1$, we can divide by $p(1-p)$, and we get

$$(x_j - x_i)^2 \leq 0,$$

which contradicts since $x_j \neq x_i$. This proves that

$$(p_i + p_j)x'_j\Delta(x'_j) > p_ix_i\Delta(x_i) + p_jx_j\Delta(x_j). \tag{2.18}$$

Now we prove that $(p_i + p_j)(1 - x'_j)\Delta(x'_j) < p_i(1 - x_i)\Delta(x_i) + p_j(1 - x_j)\Delta(x_j)$. To do so, we first show that $(p_i + p_j)\Delta(x'_j) = p_i\Delta(x_i) + p_j\Delta(x_j)$.

We know that

$$\begin{aligned}
& x'_j = \frac{p_i}{p_i + p_j}x_i + \frac{p_j}{p_i + p_j}x_j \\
\iff \quad & (p_i + p_j)x'_j = p_ix_i + p_jx_j \\
\iff \quad & (p_i + p_j)(x'_j - \frac{c}{s_1}) = p_i(x_i - \frac{c}{s_1}) + p_j(x_j - \frac{c}{s_1}) \\
\iff \quad & (p_i + p_j)s_2(x'_j - \frac{c}{s_1}) = p_is_2(x_i - \frac{c}{s_1}) + p_js_2(x_j - \frac{c}{s_1}) \\
\iff \quad & (p_i + p_j)(1 - s_2(x'_j - \frac{c}{s_1})) = p_i(1 - s_2(x_i - \frac{c}{s_1})) + p_j(1 - s_2(x_j - \frac{c}{s_1}))
\end{aligned}$$

Recall that $\Delta(x) = 1 - s_2(x - \frac{c}{s_1})$ when $\frac{c}{s_1} \leq x \leq 1$. We know that $\frac{c}{s_1} \leq x_i, x_j, x'_j \leq 1$. Thus, we get

$$(p_i + p_j)\Delta(x'_j) = p_i\Delta(x_i) + p_j\Delta(x_j). \tag{2.19}$$

By subtracting (2.18) from (2.19), we obtain

$$(p_i + p_j)(1 - x'_j)\Delta(x'_j) < p_i(1 - x_i)\Delta(x_i) + p_j(1 - x_j)\Delta(x_j).$$

□

Lemma 2.5.17. *For any (c, s_1, s_2)-El Farol game, $BCE_{\geq 2}$ has at most one configuration, $C(x)$, where $x \geq \frac{c}{s_1}$.*

Proof. Assume by way of contradiction that $BCE_{\geq 2}$ has at least two configurations that have fractions of players advised to go such that these fractions are at least $\frac{c}{s_1}$. Now let M_k be a mediator that uses $BCE_{\geq 2} = \mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_i), p_i), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $x_j > x_i \geq \frac{c}{s_1}$.

By Fact 2.5.4, we have

$$p_i x_i \Delta(x_i) + p_j x_j \Delta(x_j) + \sum_{1 \leq r \leq k, r \neq i, r \neq j} p_r x_r \Delta(x_r) \geq 0 \tag{2.20}$$

and

$$p_i(1 - x_i)\Delta(x_i) + p_j(1 - x_j)\Delta(x_j) + \sum_{1 \leq r \leq k, r \neq i, r \neq j} p_r(1 - x_r)\Delta(x_r) \leq 0. \tag{2.21}$$

Let $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_{i-1}), p_{i-1}), (C(x_{i+1}), p_{i+1}), \ldots, (C(x'_j), p_i + p_j), \ldots, (C(x_k), p_k)\}$ be a configuration distribution that has $x'_j = \frac{p_i}{p_i + p_j} x_i + \frac{p_j}{p_i + p_j} x_j$. By Fact 2.5.16, we have

$$(p_i + p_j)x'_j\Delta(x'_j) > p_i x_i \Delta(x_i) + p_j x_j \Delta(x_j) \tag{2.22}$$

and

$$(p_i + p_j)(1 - x'_j)\Delta(x'_j) < p_i(1 - x_i)\Delta(x_i) + p_j(1 - x_j)\Delta(x_j). \tag{2.23}$$

By Inequalities (2.20) and (2.22), we get

$$(p_i + p_j)x'_j\Delta(x'_j) + \sum_{1\le r\le k, r\ne i, r\ne j} p_r x_r \Delta(x_r) > 0. \tag{2.24}$$

Similarly, by Inequalities (2.21) and (2.23), we obtain

$$(p_i + p_j)(1 - x'_j)\Delta(x'_j) + \sum_{1\le r\le k, r\ne i, r\ne j} p_r(1 - x_r)\Delta(x_r) < 0. \tag{2.25}$$

By Fact 2.5.4 and Inequalities (2.24) and (2.25), $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_{i-1}), p_{i-1}), (C(x_{i+1}), p_{i+1}), \ldots, (C(x'_j), p_i + p_j), \ldots, (C(x_k), p_k)\}$ is a correlated equilibrium. Let M_{k-1} be a mediator that uses this correlated equilibrium. By Fact 2.5.5, the expected social cost of M_k is

$$((1 - \sum_{1\le r\le k, r\ne i, r\ne j} p_r x_r \Delta(x_r)) - p_i x_i \Delta(x_i) - p_j x_j \Delta(x_j))n,$$

and the expected social cost of M_{k-1} is

$$((1 - \sum_{1\le r\le k, r\ne i, r\ne j} p_r x_r \Delta(x_r)) - (p_i + p_j)x'_j\Delta(x'_j))n.$$

Since $(p_i+p_j)x'_j\Delta(x'_j) > p_i x_i \Delta(x_i) + p_j x_j \Delta(x_j)$, the expected social cost of M_{k-1} is less than the expected social cost of M_k. Further, by Lemma 2.5.15, $BCE_{\ge 2}$ must have a configuration that has no players advised to go; this implies that M_k has $k \ge 3$ configurations.

Those contradict that $BCE_{\ge 2}$ has the least social cost over all correlated equilibria of at least two configurations. □

Now we show Lemma 2.5.18 after proving Lemmas 2.5.15 and 2.5.17, as illustrated in Figure 2.8. In Lemma 2.5.18, we show that $BCE_2 = BCE_{\ge 2}$. Moreover, BCE_2 has exactly two configurations: one configuration lies on the negative network effect and one configuration lies on the positive network effect.

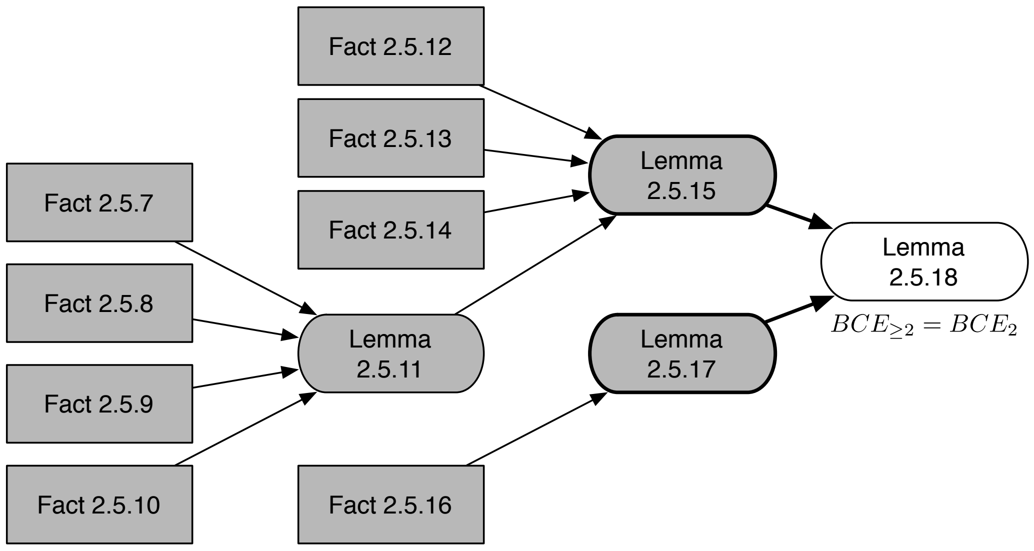

Figure 2.8: Proof Chart of Lemma 2.5.18 after proving Lemmas 2.5.15 and 2.5.17

Lemma 2.5.18. *For any* (c, s_1, s_2)*-El Farol game,* $BCE_{\geq 2} = \mathcal{D}\{(C(0), p), (C(x), 1-p)\}$*, where* $0 < p < 1$*; and* $\frac{c}{s_1} \leq x < \frac{1}{s_2} + \frac{c}{s_1}$ *if* $f(1) \geq 1$*, otherwise* $\frac{c}{s_1} \leq x \leq 1$*.*

Proof. By definition, $BCE_{\geq 2}$ has at least two configurations. By Lemmas 2.5.15 and 2.5.17, $BCE_{\geq 2}$ has exactly two configurations. The first configuration has no players advised to go, and the second configuration has an x-fraction of players advised to go, where $x \geq \frac{c}{s_1}$. We know by Fact 2.5.6 that $\Delta(0) < 0$. Thus, the first configuration has $\Delta(0) < 0$. By Fact 2.5.12, the second configuration must have $\Delta(x) > 0$. We know that $x \geq \frac{c}{s_1}$. By Fact 2.5.6, if $f(1) \geq 1$, then $\frac{c}{s_1} \leq x < \frac{1}{s_2} + \frac{c}{s_1}$; otherwise, $\frac{c}{s_1} \leq x \leq 1$. □

(B) Characterization of $BCE_{\geq 2}$

After we have proven that $BCE_{\geq 2}$ has exactly two configurations, we now characterize these configurations along with the probability distribution in terms of c, s_1 and s_2.

Lemma 2.5.19. *For any* (c, s_1, s_2)*-El Farol game, if* $c > 1$*, then* $BCE_{\geq 2}$ *is* $\mathcal{D}\{(C(0), p),$

$(C(x), 1-p)\}$, *where* $\lambda(c, s_1, s_2) = c(\frac{1}{s_1} + \frac{1}{s_2}) - \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}}$,

$$x = \begin{cases} \lambda(c, s_1, s_2) & \textit{if } \frac{c}{s_1} \leq \lambda(c, s_1, s_2) < 1, \\ \frac{c}{s_1} & \textit{if } \lambda(c, s_1, s_2) < \frac{c}{s_1}, \\ 1 & \textit{otherwise.} \end{cases}$$

and $p = \frac{(1-x)(1-f(x))}{(1-x)(1-f(x))+c-1}$.

Proof. Let M_2 be a mediator that uses $BCE_{\geq 2}$. By Lemma 2.5.18, M_2 uses $\mathcal{D}\{(C(0), p), (C(x), 1-p)\}$, where $\frac{c}{s_1} \leq x < \frac{1}{s_2} + \frac{c}{s_1}$ if $f(1) \geq 1$; otherwise, $\frac{c}{s_1} \leq x \leq 1$.

Now we determine p and x so that M_2 has the least social cost as it implements $BCE_{\geq 2}$. First, we determine p. By Constraint (2.4) of Fact 2.5.4, we have

$$p\Delta(0) + (1-p)(1-x)\Delta(x) \leq 0. \tag{2.26}$$

We know that $c > 1$, $\Delta(0) = 1 - c$ and $\Delta(x) > 0$. By rearranging Inequality (2.26), we obtain

$$p \geq \frac{(1-x)\Delta(x)}{(c-1) + (1-x)\Delta(x)}. \tag{2.27}$$

Recall that the cost of any configuration, $C(x_i)$, is $Cost(C(x_i)) = (1 - x_i\Delta(x_i))n$. Thus, we have $Cost(C(0)) = n$ and $Cost(C(x)) = (1 - x\Delta(x))n$.

Since $\Delta(x) > 0$, $Cost(C(x)) < n$. Therefore, we obtain that $Cost(C(x)) < Cost(C(0))$. We know that the expected social cost of M_2 is $pCost(C(0)) + (1-p)Cost(C(x))$. Recall that $BCE_{\geq 2}$ has the least social cost over all correlated equilibria of at least two configurations. In order to compute the social cost of M_2, given that $Cost(C(0)) > Cost(C(x))$, we compute the smallest possible value of p in Inequality (2.27). Thus, we have $p = \frac{(1-x)\Delta(x)}{(c-1)+(1-x)\Delta(x)}$.

Now we compute x of the social cost of M_2. By Fact 2.5.5, the social cost of M_2 is $(1 - (1-p)x\Delta(x))n$. Since $p = \frac{(1-x)\Delta(x)}{(c-1)+(1-x)\Delta(x)}$, the social cost of M_2 is $(1 - \frac{(c-1)x\Delta(x)}{(c-1)+(1-x)\Delta(x)})n$, or equivalently, $(1 - g(x))n$, where $g(x) = \frac{(c-1)x\Delta(x)}{(c-1)+(1-x)\Delta(x)}$.

In order to minimize the social cost with respect to x, we maximize $g(x)$ with respect to x. Hence, we have

$$\frac{dg(x)}{dx} = \frac{(c-1)[(c-1+(1-x)\Delta(x))(\Delta(x)-s_2x)+x\Delta(x)((1-x)s_2+\Delta(x))]}{((c-1)+(1-x)\Delta(x))^2}.$$

By rearranging and canceling common terms, we obtain

$$\frac{dg(x)}{dx} = \frac{(c-1)[(\Delta(x))^2+(c-1)\Delta(x)-(c-1)s_2x]}{((c-1)+(1-x)\Delta(x))^2}. \tag{2.28}$$

We know that $\Delta(x) > 0$, $\frac{c}{s_1} \le x < \frac{c}{s_1} + \frac{1}{s_2}$, $x \le 1$ and $c > 1$. Thus, the denominator of the right hand side of Equation (2.28) is always positive. By setting the numerator to zero and dividing by $c-1$, we get

$$(\Delta(x))^2 + (c-1)\Delta(x) - (c-1)s_2x = 0. \tag{2.29}$$

By solving Equation (2.29), we have $x = c(\frac{1}{s_1}+\frac{1}{s_2}) \pm \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}}$. Now let $\lambda(c,s_1,s_2) = c(\frac{1}{s_1}+\frac{1}{s_2}) - \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}}$ and $\bar{\lambda}(c,s_1,s_2) = (c(\frac{1}{s_1}+\frac{1}{s_2}) + \sqrt{\frac{c(\frac{1}{s_1}+\frac{1}{s_2})(c-1)}{s_2}})$.

Since $\bar{\lambda}(c,s_1,s_2) > (\frac{1}{s_2}+\frac{c}{s_1})$, by Lemma 2.5.18, it is out of range. This implies that we have one root $x = \lambda(c,s_1,s_2)$.

Recall that $g(x)$ is a quadratic function of x. Moreover, we know that $\frac{dg(x)}{dx}\mid_{(x=\frac{c}{s_1})} < 0$ iff $\lambda(c,s_1,s_2) < \frac{c}{s_1}$, and $\frac{dg(x)}{dx}\mid_{(x=1)} > 0$ iff $\lambda(c,s_1,s_2) > 1$. Further, we know that $\lambda(c,s_1,s_2) < \frac{c}{s_1} + \frac{1}{s_2}$; and if $f(1) \ge 1$, then $\frac{c}{s_1} \le x < \frac{1}{s_2} + \frac{c}{s_1}$, otherwise, $\frac{c}{s_1} \le x \le 1$.

Thus, for $\frac{c}{s_1} \le \lambda(c,s_1,s_2) \le 1$, the maximum of $g(x)$ is at $x = \lambda(c,s_1,s_2)$; for $\lambda(c,s_1,s_2) < \frac{c}{s_1}$, the maximum of $g(x)$ is at $x = \frac{c}{s_1}$; and for $\lambda(c,s_1,s_2) > 1$, the maximum of $g(x)$ is at $x = 1$. □

(C) $BCE_{\ge 2} \le BCE_1$

We have proven that $BCE_{\ge 2} = BCE_2$. Thus, for any (c,s_1,s_2)-El Farol game, if $c > 1$, then BCE is either BCE_1 or BCE_2. Now we show that BCE_2 has social cost of at most the social cost of BCE_1.

Recall that the social cost of BCE_2 is $(1 - g(x^*))n$, where $g(x) = \frac{(c-1)x\Delta(x)}{(c-1)+(1-x)\Delta(x)}$.

For $f(1) > 1$, by Lemma 2.4.2, the social cost of BCE_1 is n. Since $g(x^*) \geq 0$, $(1-g(x^*)) \leq 1$. Thus, the social cost of BCE_2 is at most the social cost of BCE_1.

For $f(1) \leq 1$, by Lemma 2.4.2, the social cost of BCE_1 is $f(1)n$. Since $g(x)$ is maximized at $x = x^*$, $g(x^*) \geq g(1)$. We know that $g(1) = \Delta(1)$. Thus, $g(x^*) \geq \Delta(1)$, or equivalently, $1 - g(x^*) \leq f(1)$. This implies that the social cost of BCE_2 is at most the social cost of BCE_1.

Moreover, we know that if $x^* \to 1$, then $p \to 0$. In this case, BCE_2 reduces to BCE_1, i.e., the optimal mediator implements the best Nash equilibrium, where all players are advised to go.

2.5.2 Optimal Mediator for $c \leq 1$

The following lemma shows that if $c \leq 1$, then the optimal mediator implements the best Nash equilibrium.

Lemma 2.5.20. *For any (c, s_1, s_2)-El Farol game, if $c \leq 1$, then the best correlated equilibrium is the best Nash equilibrium.*

Proof. First, in a manner similar to Fact 2.5.12, any optimal mediator over $k \geq 2$ does not have a configuration $C(x)$ with $\Delta(x) = 0$. Second, in a manner similar to Lemma 2.5.11, for any correlated, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x_j), p_j), \ldots, (C(x_k), p_k)\}$, where $0 \leq x_j < \frac{c}{s_1}$, there exists a correlated equilibrium, $\mathcal{D}\{(C(x_1), p_1), \ldots, (C(x'_j), p_j), \ldots, (C(x_k), p_k)\}$, with less expected social cost, where $\frac{c}{s_1} \leq x'_j < \frac{1}{s_2} + \frac{c}{s_1}$ if $f(1) \geq 1$; otherwise, $\frac{c}{s_1} \leq x'_j \leq 1$. Third, in a manner similar to Lemma 2.5.17, any optimal mediator has at most one configuration, $C(x)$, where $x \geq \frac{c}{s_1}$.

Therefore, for $c \leq 1$, the optimal mediator uses a configuration distribution of at most one configuration. This implies that the best correlated equilibrium is a configuration distribution of one configuration, which is trivially the best Nash equilibrium that is stated in Lemma 2.4.2. □

2.5.3 Mediation Metrics

Now we compute the *Mediation Value* and the *Enforcement Value*. Recall that the *Mediation Value* (MV) is the ratio of the minimum social cost over all Nash equilibria to the minimum social cost over all mediators; and the *Enforcement Value* is the ratio of the minimum social cost over all mediators to the optimal social cost.

For $c > 1$, by Lemmas 2.4.2 and 2.5.19,

$$MV = \frac{\min(f(1), 1)}{p + (1-p)(xf(x) + (1-x))};$$

and by Lemmas 2.4.1 and 2.5.19,

$$EV = \frac{p + (1-p)(xf(x) + (1-x))}{yf(y) + (1-y)}.$$

For $c \leq 1$, by Lemma 2.5.20,

$$MV = 1;$$

and by Lemmas 2.4.1 and 2.4.2,

$$EV = \frac{\min(f(1), 1)}{yf(y) + (1-y)}.$$

Chapter 3

Self-Healing Communication

"Fool me once, shame on you. Fool me twice, shame on me."
– Randall Terry

3.1 Introduction

Self-healing algorithms protect critical properties of a network, even when that network is under repeated attack. Such algorithms detect corruptions efficiently and expend resources when it is necessary to repair damage done by an attacker. Thus, they provide significant resource savings when compared to traditional robust algorithms, which expend significant resources even when the network is not under attack.

The last several years have seen exciting results in the design of self-healing algorithms [19, 37, 38, 64, 73, 75]. Unfortunately, none of these previous results handle *Byzantine faults*, where an adversary takes over nodes in the network and can cause them to deviate arbitrarily from the protocol. This is a significant gap, since traditional Byzantine-resilient algorithms are notoriously inefficient, and the self-healing approach could significantly improve efficiency.

In this chapter, we take a step towards addressing this gap. For a network of n nodes, we design self-healing algorithms for communication that tolerate up to $1/4$ fraction of Byzantine faults. Our algorithms enable any node to send a message to any other node in the network with message and latency costs that are asymptotically optimal.

Moreover, our algorithms limit the expected total number of message corruptions. Ideally, each Byzantine node would cause $O(1)$ corruptions; our result is that each Byzantine node causes $O((\log^* n)^2)$ corruptions in expectation. [1] [2]

This chapter is organized as follows. In Section 3.2, we describe our model. Our main theorem is given in Section 3.3, and we provide a technical overview in Section 3.4. The related work is discussed in Section 3.5. Section 3.6 describes our algorithms. The analysis of our algorithms is shown in Section 3.7. Section 3.8 gives empirical results showing how our algorithms improve the efficiency of the butterfly networks of [30].

3.2 Our Model

We assume a *static* Byzantine adversary in the sense that it takes over nodes before the algorithm begins. The nodes that are controlled by the adversary are *bad*, and the other nodes are *good*. The bad nodes may arbitrarily deviate from the protocol, by sending no messages, excessive numbers of messages, incorrect messages, or any combination of these. The good nodes follow the protocol.

Further, we assume that the adversary knows our protocol, but is unaware of the random bits of the good nodes. We assume that each node has a unique ID. We say that node p has a link to node q if p knows q's ID and can thus directly communicate with node q. Also, we assume the existence of a public key digital signature scheme, and thus a computationally

[1] Recall that $\log^* n$ or the iterated logarithm function is the number of times logarithm must be applied iteratively before the result is less than or equal to 1. It is an extremely slowly growing function: e.g. $\log^* 10^{10} = 5$.

[2] We thus amend our initial quote to: *"Fool me once, shame on you. Fool me $\omega((\log^* n)^2)$ times, shame on me."*

bounded adversary. Also, we assume that the network remains physically connected even after the adversary removes all Byzantine nodes.

Moreover, we assume partially synchronous communication model in the sense that any message sent from one good node to another good node requires at most h time steps to be sent and received, and the value h is known to all nodes. Also, we tolerate if the adversary is *rushing*, where the bad nodes receive all messages from good nodes in a round before sending out their own messages.

3.3 Our Contributions

This chapter provides a self-healing algorithm, *SEND*, that sends a message reliably from a source node to a target node through a network. Our main theoretical result is summarized in the following theorem.

Theorem 3.3.1. *Assume we have a network with n nodes and $t \leq (1/4 - \epsilon)n$ bad nodes, for any constant $\epsilon > 0$. Then our algorithm has the following properties:*

(1) in an amortized sense[3]*, any call to SEND, to deliver a message reliably through a path of length ℓ, has expected number of messages $O(\ell + \log n)$ with expected latency $O(\ell)$; and*

(2) the expected total number of times that SEND fails to deliver a message reliably is $O(t(\log^* n)^2)$.

Our experimental results (Section 3.8) show that our algorithms reduce the message cost, compared to a naive algorithm that has no self-healing properties, by a factor of 50 for

[3]In particular, if we call *SEND* $\mathcal{L}$ times through quorum paths, where ℓ_M is the longest such path, then the expected total number of messages sent will be $O(\mathcal{L}(\ell_M + \log n) + t \cdot (\ell_M \log^2 n + \log^5 n))$ with latency $O(\ell(\mathcal{L} + t))$. Since t is fixed for large $\mathcal{L}$, the expected number of messages per *SEND* is $O(\ell_M + \log n)$ with expected latency $O(\ell)$.

$n = 14{,}116$, and by a factor of 60 for $n = 30{,}509$.

3.4 Technical Overview

In this section, we describe quorum graph, naive communication through the quorum graph and our self-healing approach for communication networks.

3.4.1 Quorum Graph of Communication Network

We define a *quorum* to be a set of $\Theta(\log n)$ nodes, of which at most 1/4-fraction are bad. Many results show how to create and maintain a network of quorums [14, 30, 31, 39, 46, 60, 76]. All of these results maintain what we will call a *quorum graph* in which each vertex represents a quorum. The properties of the quorum graph are:

(1) each node is in $O(\log n)$ quorums;

(2) for any quorum Q, any node in Q can communicate directly to any other node in Q; and

(3) for any quorums Q_i and Q_j that are connected in the quorum graph, any node in Q_i can communicate directly with any node in Q_j and vice versa.

Moreover, we assume that for any two nodes x and y in a quorum, node x knows all quorums that node y is in.

Note that the quorum graph is created after the adversary chooses the bad nodes.

3.4.2 Communicating with Quorums

The communication in the quorum graph typically occurs as follows. When a node **s** sends another node **r** some message m, there is a canonical *quorum path*, $Q_1, Q_2, \ldots, Q_\ell$, through the quorum graph. This path is determined by the ID's of both **s** and **r**. Note that we assume that node **s** is a good node.

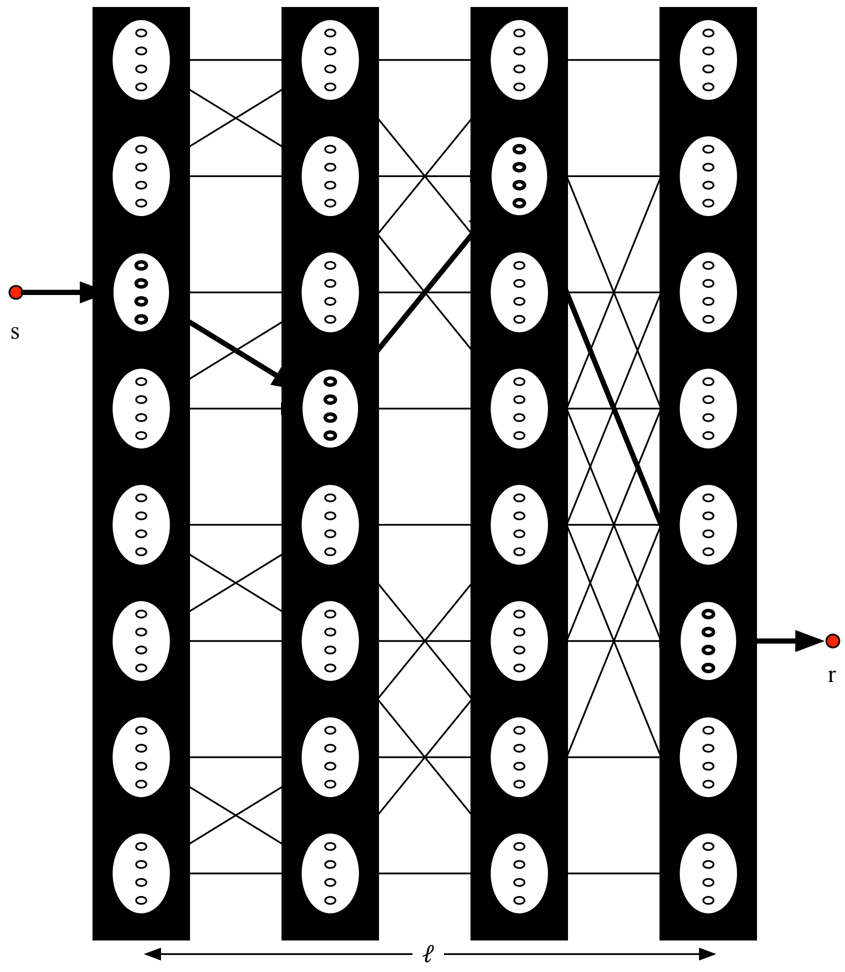

Figure 3.1: Quorum Graph of Communication Network

Figure 3.1 shows the communication between node **s** and node **r** through a quorum path in the quorum graph, where the message is propagated from the left to the right.

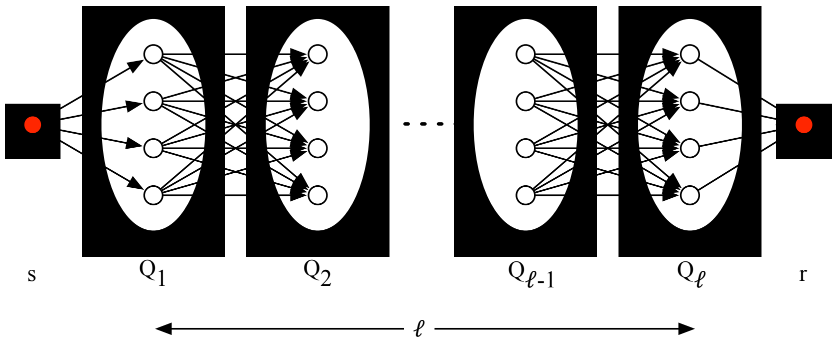

Figure 3.2: Naive Communication

3.4.3 Naive Communication

A correct but inefficient algorithm to route a message m reliably from a source node to a target node is shown in Figure 3.2. In this algorithm, node **s** sends a message m to node **r** through a path of quorums via all-to-all communication. Each node participating in the quorum path takes the majority of the messages it receives in order to determine the true value of m.

Unfortunately, this algorithm requires $O(\ell \log^2 n)$ messages and latency $O(\ell)$. A main result of this chapter is to reduce the message cost to $O(\ell + \log n)$ in an amortized sense.

3.4.4 Our Approach

An efficient approach to communication is to have a path of nodes selected uniformly at random through the quorum path from node **s** to node **r**. In this path of nodes, each node forwards the message it has received to eventually be received by node **r**. Unfortunately, a single bad node in this path can corrupt the entire communication.

We provide an algorithm, *CHECK*, which detects if there has been a corruption. *CHECK* is a light-weight algorithm in terms of message cost. Node **s** triggers *CHECK* with some probability. If *CHECK* detects a corruption, it calls *HEAL*, which is a heavy-weight algorithm but it is bounded in an amortized sense. In particular, the expected total number of calls to *HEAL* before all bad nodes are marked is $O(t)$.

CHECK is implemented as either *CHECK1* or *CHECK2*. *CHECK1* runs in one round. Node **s** resends the message through a path of subsets of $2 \log \log n$ nodes in the quorum path via all-to-all communication. *CHECK1* fails to detect corruptions if all nodes in any subset are bad. A key lemma (Lemma 3.7.1) shows that this algorithm fails to detect a corruption with probability less than 1/2 for $\ell \leq \frac{\log^2 n}{2}$.

CHECK2 is a more sophisticated algorithm. It runs in $O(\log^* n)$ rounds. In each round, node **s** sends the message through a path of subsets of nodes in the quorum path. These subsets are initially empty. In each round, a new node selected uniformly at random is added to each subset.

In order to show how *CHECK2* detects a corruption, we first define *deception interval in a round* as a path of bad nodes that are selected in this round to be added to the subsets. Note that the adversary selects the deception intervals, in which the bad nodes corrupt the messages. Note further that if the adversary corrupts a message in any round, it has to keep corrupting this message in all subsequent rounds. Thus, for i rounds of *CHECK2* to fail to detect a corruption, there must be nesting levels of deception intervals in each of those i rounds. A key lemma (Lemma 3.7.4) shows that any deception interval shrinks logarithmically from round to round with probability at least 1/2. We use this lemma to show that *CHECK2* requires $O(\log^* n)$ rounds to detect corruption with constant probability.[4]

In *HEAL*, each node has participated in communication is investigated in order to determine which node(s) has corrupted the message. In particular, each node announces the messages it has received. In this way, each call to *HEAL* identifies at least one pair of nodes that are in conflict; informally, we say that a pair of nodes are in conflict if they each accuse

[4]This probability can be made arbitrarily close to 1 by adjusting the hidden constant in the $O(\log^* n)$ rounds. See Corollary 3.7.10.

the other of malicious behavior. In such a situation, we know that at least one node in this pair is bad. In *HEAL*, both nodes in each conflicting pair are marked, and these marked nodes are prohibited from participating in future communication.

A naive approach would be to never unmark marked nodes. Unfortunately, this approach fails because all nodes in a quorum may get marked and so inhibit communication. To avoid this, we unmark the nodes of any quorum that has $1/2$-fraction of nodes marked. A subtle potential function argument (Lemma 3.7.12) shows that this marking scheme will mark all bad nodes after $O(t)$ calls to *HEAL*, after which the network is completely healed, i.e., no more corruption will occur.

3.4.5 Marked Senders

We can allow marked nodes to call *SEND*, where it may be helpful to let the good nodes that are marked still be able to lookup. However, the marked bad nodes may blow up the message cost of *SEND* by a factor of $O((\log\log n)^2)$ if *CHECK1* is used and by a factor of $O((\log^* n)^2)$ if *CHECK2* is used.

3.5 Related Work

Several papers [35, 40, 57, 81, 82] have discussed different restoration mechanisms to preserve network performance by adding capacity and rerouting traffic streams in the presence of node or link failures. They present mathematical models to determine global optimal restoration paths, and provide methods for capacity optimization of path-restorable networks.

Our results are inspired by recent work on self-healing algorithms [19, 37, 38, 64, 73, 75]. A common model for these results is that the following process repeats indefinitely: an adversary deletes some nodes in the network, and the algorithm adds edges. The algorithm is constrained to never increase the degree of any node by more than a logarithmic factor from its original degree. In this model, researchers have presented algorithms that ensure the following properties: the network stays connected and the diameter does not increase by much [19, 37, 73]; the shortest path between any pair of nodes does not increase by much [38]; and expansion properties of the network are approximately preserved [64].

Our results are also similar in spirit to those of Saia and Young [74] and Young et al. [84], which both show how to reduce message complexity when transmitting a message across a quorum path of length ℓ. The first result, [74], achieves expected message complexity of $O(\ell \log n)$ by use of bipartite expanders. However, this result is impractical due to high hidden constants and high setup costs. The second result, [84], achieves expected message complexity of $O(\ell)$. However, this second result requires the sender to iteratively contact a member of each quorum in the quorum path.

As mentioned earlier, several peer-to-peer networks have been described that provably enable reliable communication, even in the face of adversarial attack [12, 13, 14, 25, 30, 39, 43, 60, 76]. To the best of our knowledge, our approach applies to each of these networks, with the exception of [25]. In particular, we can apply our algorithms to asymptotically improve the efficiency of the peer-to-peer networks from [14, 30, 39, 60, 76].

Similar to Young et al. [85], we use threshold cryptography as an alternative to Byzantine Agreement.

In this chapter, we improve the results of [48] to tolerate up to $t \leq (1/4 - \epsilon)n$ bad nodes, for any constant $\epsilon > 0$, instead of $t \leq (1/8 - \epsilon)n$ bad nodes.

3.6 Our Algorithms

In this section, we describe our algorithms: *SEND*, *SEND-PATH*, *CHECK* and *HEAL*. The main technical challenge of this chapter is in the design of the algorithm *CHECK*, which is described in Section 3.6.4.

3.6.1 Overview

The objective of our algorithms is to detect the corruption and to mark all bad nodes in the network, after which no message corruption occurs. If a node is marked, it is not allowed to participate in communication. Before our algorithms start, all nodes in the network are initially unmarked.

Before discussing our main *SEND* algorithm, we describe that when a node x broadcasts a message m, signed by the private key of a quorum Q, to a set of nodes S, it calls *BROADCAST* (m, Q, S).

3.6.2 *BROADCAST*

In *BROADCAST* (Algorithm 1), we use threshold cryptography to avoid the overhead of Byzantine Agreement.

In a (η, η')-threshold cryptographic scheme, a private key is distributed among η nodes in such a way that 1) any subset of more than η' nodes can jointly reassemble the key; and 2) no subset of at most η' nodes can recover the key. The private key can be distributed using a *Distributed Key Generation* (DKG) protocol [45].

DKG generates the public/private key shares of all nodes in every quorum. We assume that the public key share of each node and the public key of each quorum are known to all nodes in the quorum and the neighboring quorums in the network.

In particular, we use a $(|Q|, \frac{3|Q|}{4} - 1)$-threshold scheme, where $|Q|$ is the quorum size. A node x calls *BROADCAST* in order to send a message m to all nodes in S so that: 1) at least 3/4-fraction of the nodes in quorum Q have received the same message m; 2) they

Algorithm 1 BROADCAST(m, Q, S) $\triangleright$ A node x sends a message m, signed by quorum Q, to a set of nodes S.

```
1: Node $x$ sends message $m$ to all nodes in $Q$.
2: Each node in $Q$ signs $m$ by its private key share to obtain its message share.
3: Each node in $Q$ sends its message share back to node $x$.
4: Node $x$ interpolates at least $\frac{3|Q|}{4}$ message shares to obtain a signed-message of $Q$.
5: Node $x$ sends this signed-message to all nodes in $S$.
```

agree upon the content of m; and 3) they give a permission to x to broadcast this message.

Any call to *BROADCAST* requires $O(\log n + |S|)$ messages for signing the message m by $O(\log n)$ nodes in quorum Q, with latency $O(1)$.

3.6.3 *SEND*

Now we describe our main algorithm, *SEND*, that is stated formally in Algorithm 2. *SEND* calls *SEND-PATH*, which is described in Algorithm 3.

Algorithm 2 SEND$(m, \mathbf{r})$ $\triangleright$ node $\mathbf{s}$ sends a message m reliably to node $\mathbf{r}$.

```
1: Node $\mathbf{s}$ calls SEND-PATH $(m, \mathbf{r})$.
2: With probability $p_c$, node $\mathbf{s}$ calls CHECK $(m, \mathbf{r})$.
```

In *SEND-PATH*, as shown in Figure 3.3, node $\mathbf{s}$ sends message m to node $\mathbf{r}$ through a path of unmarked nodes selected uniformly at random. It is efficient in terms of message cost and latency; but in the presence of bad nodes, it is vulnerable to corruption. Thus, we make *SEND* call *CHECK* algorithm with probability p_c, where *CHECK* detects, with probability p_d, if a message has been corrupted in the last call to *SEND-PATH*.

In *CHECK*, the message is sent from node $\mathbf{s}$ to node $\mathbf{r}$ through a path of subquorums, where a subquorum is a subset of unmarked nodes selected uniformly at random in a quorum. Unfortunately, while *CHECK* can determine if a corruption occurred, it does not specify the

Algorithm 3 SEND-PATH$(m, \mathbf{r})$ $\triangleright$ node **s** sends a message m through a path of unmarked nodes to node **r**.

Declaration: for $1 < i < \ell$, let U_i be the set of all unmarked nodes in Q_i. Also let w be the maximum number of nodes in any quorum.

```
1: Node $\mathbf{s}$ sets $R$ to be an array of $w$ integers selected uniformly at random between 1 and $w$.
2: Node $\mathbf{s}$ broadcasts $m$ and $R$ to all nodes in $Q_1$.
3: The nodes in $Q_1$ calculate the node $q_2$ using $R[|U_2|]$ to index $U_2$'s nodes sorted by their IDs.
4: All nodes in $Q_1$ send $m$ to node $q_2$.
5: for $i = 2, \ldots, \ell - 2$ do
6:     Node $q_i$ selects $q_{i+1} \in U_{i+1}$ uniformly at random.
7:     Node $q_i$ sends $m$ to node $q_{i+1}$.
8: end for
9: Node $q_{\ell-1} \in U_{\ell-1}$ broadcasts $m$ to all nodes in $Q_\ell$.
10: All nodes in $Q_\ell$ send $m$ to node $\mathbf{r}$.
```

location where the corruption occurred. Hence, if *CHECK* detects a corruption, *HEAL* algorithm is called.

When *HEAL* is called, it identifies two neighboring quorums Q_i and Q_{i+1} in the path, for some $1 \leq i < \ell$, such that at least one pair of nodes in these quorums is in conflict and

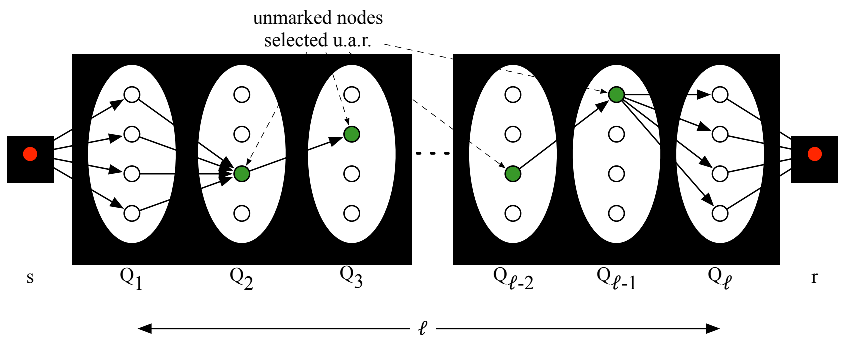

Figure 3.3: SEND-PATH Algorithm

at least one node in such pair is bad. These nodes are marked in all quorums they are in and in their neighboring quorums. Moreover, in each call to *HEAL*, we mark at most one good node. In order to always provide unmarked nodes to participate in communication, we set the following condition. If $(1/2 - \gamma)$-fraction of nodes in any quorum have been marked, for a constant $\gamma > 0$, these nodes are set unmarked. Furthermore, in order to handle any accusation against node **s** or node **r** in *HEAL*, we let the message be broadcasted to Q_1 and Q_ℓ in each call to *SEND-PATH* and *CHECK*.

Our model does not directly consider concurrency. In a real system, concurrent calls to *HEAL* that overlap at a single quorum may allow the adversary to achieve multiple corruptions at the cost of a single marked bad node. However, this does not effect correctness, and, in practice, this issue can be avoided by serializing concurrent calls to *SEND*. For simplicity of presentation, we leave the concurrency aspect out of this research.

3.6.4 *CHECK*

We implement *CHECK* as either *CHECK1* or *CHECK2*. In this section, we describe *CHECK1* and *CHECK2*. Then, we compare between them in terms of message cost and latency.

Throughout this section, we let U_j be the set of all unmarked nodes in Q_j for $1 < j < \ell$.

CHECK1

Now we describe *CHECK1* that is stated formally as Algorithm 4. *CHECK1* is a simpler *CHECK* procedure compared to *CHECK2*. Although *CHECK1* has a worse asymptotic message cost, it performs well in practice.

CHECK1 is triggered by *SEND* with probability $p_c = 1/(\log\log n)^2$. *CHECK1* runs in one round, in which node **s** sends a message m to node **r** through a path of subquorums via all-to-all communication, see Figure 3.4. Note that each subquorum, S_j, has $2 \log \log n$ nodes that are chosen uniformly at random with replacement from the nodes of U_j in the quorum path, for $1 < j < \ell$.

If any good node receives inconsistent messages or fails to receive an expected message during *CHECK1*, it initiates a call to *HEAL*. However, *CHECK1* fails to detect message corruptions if all nodes of any subquorum in the quorum path are bad.

Algorithm 4 CHECK1($m, \mathbf{r}$) ▷ checks in one round if there has been a corruption

Declaration: for $1 < j < \ell$, let U_j be the set of all unmarked nodes in Q_j and let each subquorum S_j be initially empty. Note that each subquorum will have at most $2 \log \log n$ nodes. Also, let w be the maximum number of nodes in any quorum.

```
 1: Node $\mathbf{s}$ generates $R$ as an $\ell$ by $w$ by $2 \log \log n$ array of random integers.*
 2: Node $\mathbf{s}$ sets $m'$ to be a message consisting of $m$, $\mathbf{r}$, and $R$.
 3: Node $\mathbf{s}$ broadcasts $m'$ to all nodes in $Q_1$.
 4: The nodes in $Q_1$ use $R_2$ to calculate the nodes of $S_2$.**
 5: The nodes in $Q_1$ send $m'$ to the nodes of $S_2$.
 6: for $j \leftarrow 2, \ldots, \ell - 2$ do
 7:     The nodes of $S_j$ use $R_{j+1}$ to calculate the nodes of $S_{j+1}$.
 8:     The nodes of $S_j$ send $m'$ to all nodes of $S_{j+1}$.
 9: end for
10: The nodes of $S_{\ell-1}$ broadcast $m'$ to all nodes in $Q_\ell$.
11: The nodes of $Q_\ell$ send $m'$ to node $\mathbf{r}$.
```

* $R[j, k]$ is a multiset of $2 \log \log n$ integers selected uniformly at random with replacement between 1 and k, for $1 < j < \ell$ and $1 \leq k \leq w$.

** $R[j, |U_j|]$, shortly R_j, has the indices of the nodes of S_j selected u.a.r. from the nodes of U_j; note that the nodes of U_j are sorted by their IDs.

Note that: if a node receives inconsistent messages or fails to receive an expected message, then it initiates a call to *HEAL*.

In Section 3.7, we show that if the message was corrupted during the last call to *SEND-PATH*, the probability that *CHECK1* fails to detect a corruption is less than 1/2 for $\ell \leq \frac{\log^2 n}{2}$. Hence, *CHECK1* detects message corruptions with probability $p_d > 1/2$. This requires $O(\ell(\log\log n)^2 + \log n \cdot \log\log n)$ messages and latency $O(\ell)$. But since *CHECK1* is triggered with probability $1/(\log\log n)^2$, it has expected message cost $O(\ell + \log n/\log\log n)$, with latency $O(\ell/(\log\log n)^2)$.

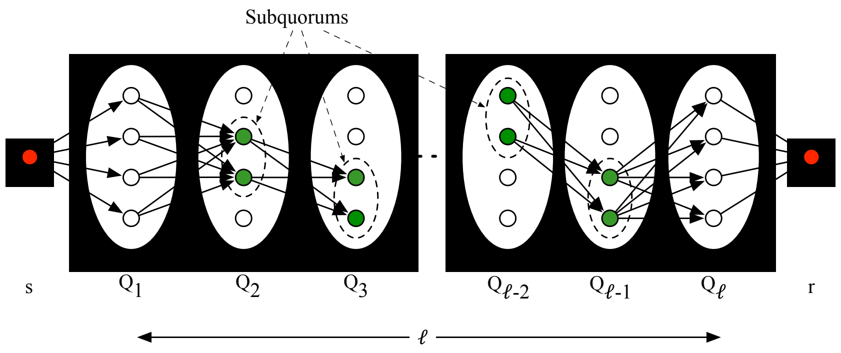

Figure 3.4: *CHECK1* Algorithm

CHECK2

In this section, we describe *CHECK2*, which is stated formally as Algorithm 5. Note that *SEND* calls *CHECK2* with probability $p_c = 1/(\log^* n)^2$, and *CHECK2* runs in $5(\log^* n + 3)$ rounds.

In *CHECK2*, firstly, node **s** generates a public/private key pair (k_p, k_s) to let the nodes that receive k_p verify any subsequent message signed by k_s. As illustrated in Figure 3.5, node **s** sends, in each round, a message m through a path of subquorums to node **r**. Note that for each call to *CHECK2*, each subquorum, S_j, in the quorum path is initially empty, and in each round, a new node $x_j \in U_j$ selected uniformly at random is added to S_j, for all $1 < j < \ell$.

Algorithm 5 CHECK2($m, \mathbf{r}$) ▷ checks in multiple rounds if there has been a corruption

Declaration: for $1 < j < \ell$, let U_j be the set of all unmarked nodes in Q_j and let each subquorum S_j be initially empty. Also let w be the maximum number of nodes in any quorum.

```
 1: Node $\mathbf{s}$ generates public/private key pair $(k_p, k_s)$.
 2: for $i \leftarrow 1, \ldots, 5(\log^* n + 3)$ do
 3:     Node $\mathbf{s}$ generates $R_i$ as an $\ell$ by $w$ array of random integers.*
 4:     Node $\mathbf{s}$ sets $m_i = ([m, \mathbf{r}, i, R_i]_{k_s}, k_p)$.
 5:     Node $\mathbf{s}$ broadcasts $m_i$ to all nodes in $Q_1$.
 6:     All nodes in $Q_1$ use $R_{i2}$ to calculate node $x_{i2}$ to be added to $S_2$.**
 7:     The nodes in $Q_1$ send $m_i$ to all nodes in $S_2$.
 8:     The nodes in $Q_1$ send $R_1, \ldots, R_{i-1}$ to node $x_{i2}$.
 9:     for $j \leftarrow 2, \ldots, \ell - 2$ do
10:         All $i$ nodes in $S_j$ use $R_{i(j+1)}$ to calculate node $x_{i(j+1)}$ to be added to $S_{j+1}$.
11:         for $k \leftarrow 1, \ldots, i - 1$ do
12:             Node $x_{kj}$ sends $m_k$ to node $x_{i(j+1)}$.
13:             Node $x_{ij}$ uses $R_{k(j+1)}$ to calculate node $x_{k(j+1)}$.
14:             Node $x_{ij}$ sends $m_i$ to node $x_{k(j+1)}$.
15:         end for
16:         Node $x_{ij}$ sends $m_i$ to node $x_{i(j+1)}$.
17:     end for
18:     The nodes in $S_{\ell-1}$ broadcast $m_i$ to all nodes in $Q_\ell$.
19:     All nodes in $Q_\ell$ send $m_i$ to node $\mathbf{r}$.
20: end for
```

* $R_i[j, k]$ is a uniformly random integer between 1 and k, for $1 < j < \ell$ and $1 \leq k \leq w$.

** $R_i[j, |U_j|]$, shortly R_{ij}, is the index of node, x_{ij}, to be selected u.a.r. from the nodes of U_j in round i; note that the nodes of U_j are sorted by their IDs.

Note that: if a node has previously received k_p, then it verifies each subsequent message with it; also if a node receives inconsistent messages or fails to receive and verify an expected message, then it initiates a call to *HEAL*.

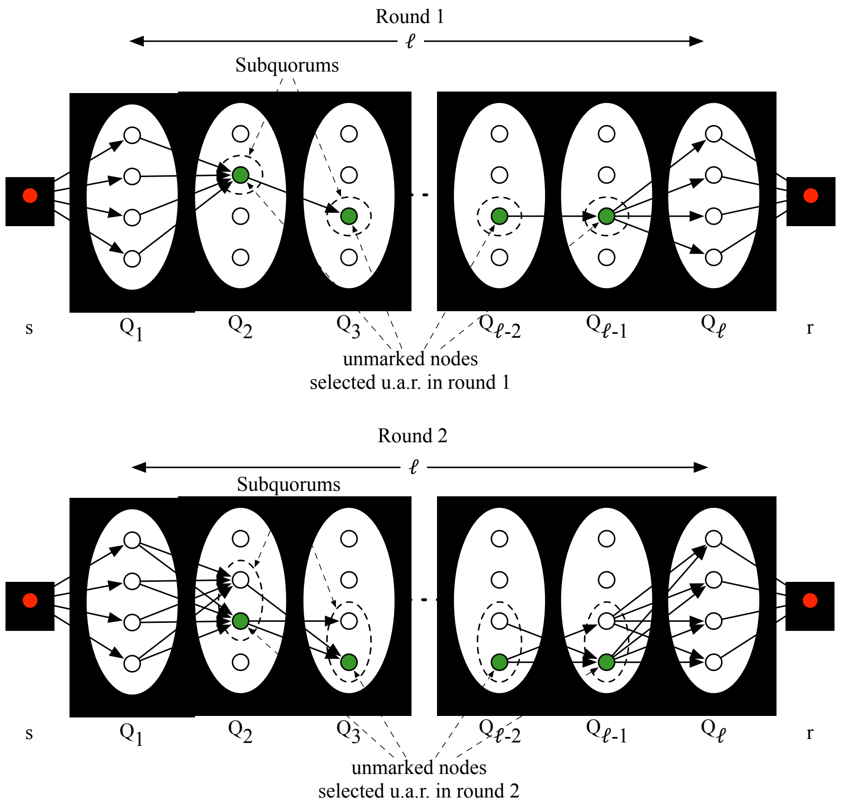

Figure 3.5: *CHECK2* Algorithm

If any node receives inconsistent messages or fails to receive and verify any expected message in any round, it initiates a call to *HEAL*. *CHECK2* detects message corruptions with probability $p_d \geq 1/2$. It requires $O((\ell + \log n)(\log^* n)^2)$ message cost and $O(\ell \log^* n)$ latency. But since *CHECK2* is triggered with probability $1/(\log^* n)^2$, it has expected message cost $O(\ell + \log n))$ with expected latency $O(\ell/ \log^* n)$.

An example run of *CHECK2* is illustrated in Figure 3.6. In this figure, there is a column for each subquorum from S_2 to $S_{\ell-1}$ in the quorum path and a row for each round of *CHECK2*. For a given row and column, there is a G or B in that position depending on whether the node selected in that particular round and that particular quorum is good (G) or bad (B).

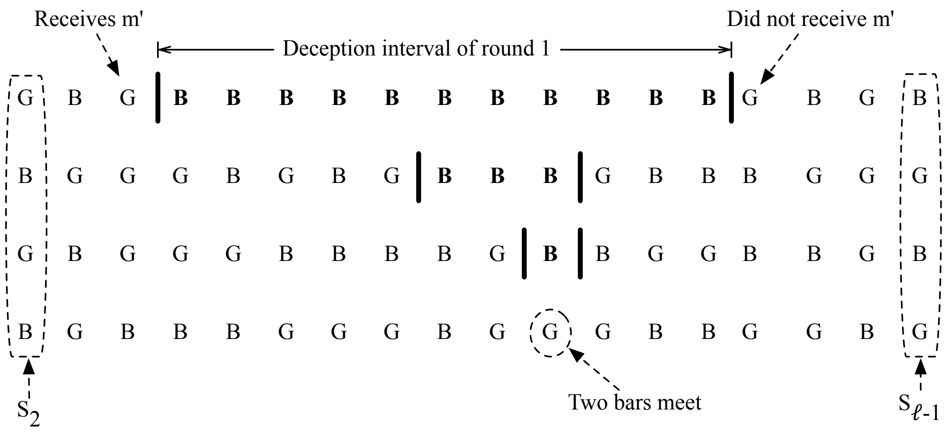

Figure 3.6: Example run of *CHECK2*

Recall that a *deception interval in a round* is a path of bad nodes that are selected in this round to be added to the subquorums in the quorum path. Note that the adversary's strategy to maximize the number of rounds before corruption detection is to select the longest deception interval in the first row, and to keep corrupting the message in all subsequent deception intervals.

In Figure 3.6, we show the deception intervals selected by the adversary. These intervals are outlined by left and right bar in each row. Note that the left bar in each row specifies the rightmost subquorum in which there is some good node that receives the correct message m'. The right bar in each row specifies the leftmost subquorum in which there is some good node that does not receive m'.

There are two key points by which *CHECK2* detects message corruptions: 1) any deception interval in any round never *expands* in any subsequent round; and 2) any deception interval *shrinks* to length zero after $O(\log^* n)$ rounds, with constant probability.

Deception intervals never expand? In order to show that any deception interval never expands over rounds, we show that the left bar never moves leftwards, and the right bar never moves rightwards.

The left bar never moves leftwards because each good node receives the message m' in round i has to receive the same message in all subsequent rounds; otherwise, it will call *HEAL*. Moreover, we know that for each round i, all nodes in each subquorum S_j send the message to the new node that is selected in this round to be added to S_{j+1} for $1 < j < \ell$. Thus, those good nodes that receive and provide m' to the deception interval in round i will provide the same message to all subsequent deception intervals.

Now we show that the right bar never moves rightwards. Recall that in each round, each node that is added to each subquorum S_j sends the message it has received to all nodes in S_{j+1} for $1 < j < \ell$. Thus, all good nodes that receive a message through a deception interval in any round expect to receive the same message in all subsequent rounds. Also, each good node that did not receive m' in any round must not receive this message in all subsequent rounds; otherwise, it will initiate a call to *HEAL*. This shows that the right bar never moves rightwards.

Deception intervals shrink logarithmically? The reason that *CHECK2* requires $O(\log^* n)$ rounds is because of a probabilistic result on the maximum length run in a sequence of coin tosses. In particular, if we have a coin that takes on value "B" with probability at most 1/2, and value "G" with probability at least 1/2, and we toss it x times, then the expected length of the longest run of B's is $O(\log x)$. Thus, if in some round, the distance between the left bar and the right bar is x, we expect in the next round this distance will shrink to $O(\log x)$. Intuitively, we might expect that, if the quorum path is of length ℓ, then $O(\log^* \ell)$ rounds will suffice before the distance shrinks to 0. This intuition is formalized in Lemma 3.7.9 (Section 3.7).

When the two bars meet, is the corruption detected? Figure 3.7 shows that when the two bars meet, a corruption is detected. In this figure, as the deception intervals shrink over rounds, node x in the last round receives message m'. Then node x forwards this message to node y which has not previously received m' in this call to *CHECK2*. As a result, node y calls *HEAL* declaring that it has received inconsistent messages.

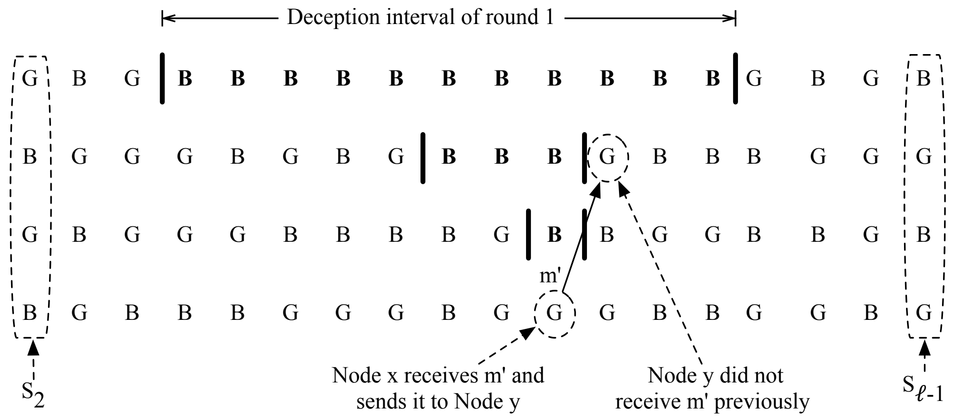
Deception interval of round 1
m'
Node x receives m' and sends it to Node y
Node y did not receive m' previously
S2
Sℓ-1

Figure 3.7: A corruption is detected after the two bars meet.

If all nodes in one or more subquorums are bad, does *CHECK2* successfully detect message corruptions? We know that *CHECK1* fails if all nodes in any subquorum in the quorum path are bad. However, *CHECK2* can detect corruptions even if all nodes in multiple subquorums are bad. Recall that *CHECK2* runs in $O(\log^* n)$ rounds. In each round, new nodes are selected uniformly at random to be added to the subquorums. This makes the adversary not be able to know, before all rounds finish, if all the nodes in one or more subquorums are bad. Thus, the adversary would rather select the longest deception interval in the first round and keeps corrupting the message in the nesting deception intervals in all subsequent rounds. Note that if the adversary corrupts the message in more than one interval in the same round, it will increase the chance of detecting message corruption.

Figure 3.8 shows that even though all nodes in a subquorum S_{10} are bad, the good node v receives message m' from the bad node u, where node u is not in the deception interval chosen by the adversary in row 3.

Furthermore, if all nodes in multiple subquorums are bad, then the corruptions that occur in the deception intervals can be detected. Figure 3.9 shows that even if all nodes in subquorums, S_5, S_{10} and S_{15}, are bad, the good node v_i receives m' from the bad node u_i,

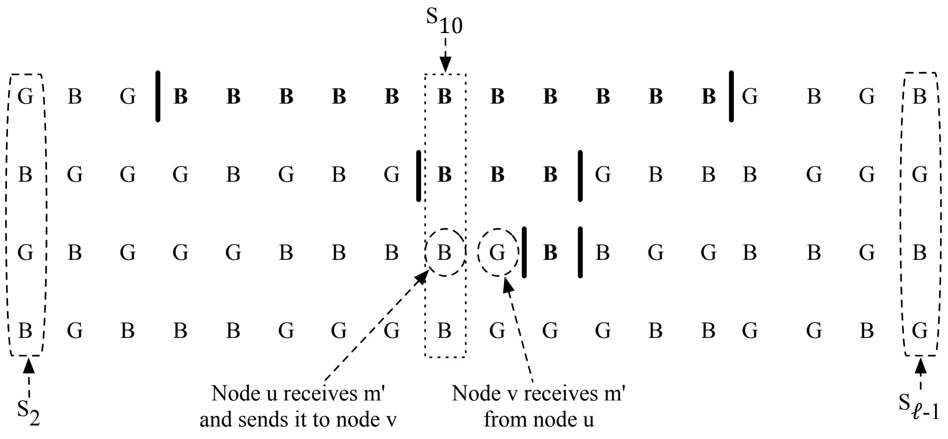
S10
Node u receives m' and sends it to node v
Node v receives m' from node u
S2
Sℓ-1

Figure 3.8: A corruption can be detected even if all nodes in subquorum S_{10} are bad.

which is not in the deception intervals, for $i \in \{5, 10, 15\}$.

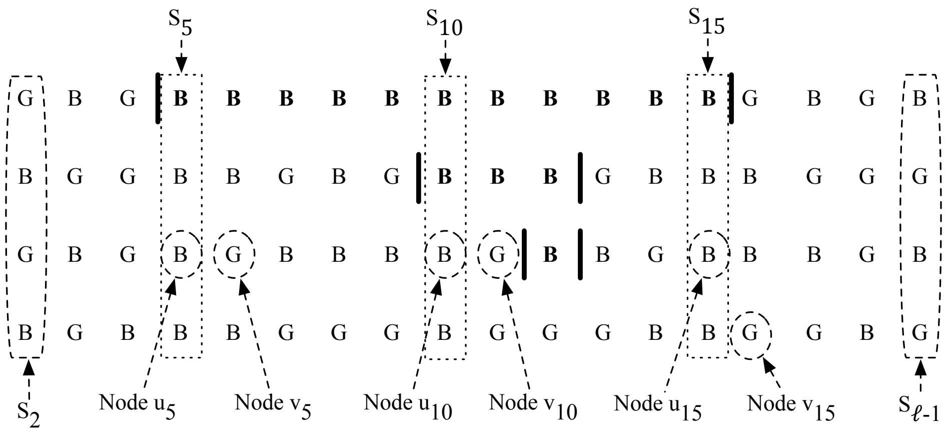

Figure 3.9: A corruption can be detected even if all nodes in multiple subquorums, S_5, S_{10} and S_{15}, are bad.

A comparison between *CHECK1* and *CHECK2*

CHECK1 has the following advantages over *CHECK2*. 1) *CHECK1* has less latency, where each call to *CHECK1* runs in one round, and each call to *CHECK2* runs in $O(\log^* n)$ rounds;

2) *CHECK1* has fewer calls to broadcast, where any call to *CHECK1* calls *BROADCAST* twice, and any call to *CHECK2* calls *BROADCAST* $O(\log^* n)$ times; and 3) *CHECK1* has less message cost in expectation, when all bad nodes are marked.

However, *CHECK2* has the following advantages over *CHECK1*. 1) *CHECK2* can handle a quorum path of length $\ell \leq n$ but *CHECK1* handles a quorum path of length $\ell \leq \frac{\log^2 n}{2}$; and 2) *CHECK2* can have less message cost initially; note that it has less message cost per round, and once it detects a corruption at any round, it terminates.

3.6.5 *HEAL*

When a message is corrupted and *CHECK* detects such a corruption, *HEAL* is called. *HEAL* is described formally as Algorithm 6.

The purpose of calling *HEAL* is 1) to determine the location at which the corruption has occurred; and 2) to mark the nodes that are in conflict.

Algorithm 6 *HEAL* ▷ Node $q' \in Q'$ calls *HEAL* after it detects a corruption.

```
1: q' broadcasts the fact that it calls HEAL along with all the messages that it has received in
   this call to SEND, to all nodes in Q'.
2: The nodes in Q' verify that q' received inconsistent messages before proceeding.
3: Q' notifies that a call to HEAL is occurring, via all-to-all communication, to all quorums in the
   quorum path.
4: INVESTIGATE
5: MARK-IN-CONFLICTS
```

When *HEAL* starts, all nodes, in each quorum in the quorum path, are notified. To determine the location at which the corruption is occurred, *HEAL* investigates the corruption situation in *INVESTIGATE*, which is stated formally as Algorithm 7. In *INVESTIGATE*, each node, previously involved in *SEND-PATH* or *CHECK*, broadcasts to all nodes in its quorum and the neighboring quorums, the messages they have received (and from whom) and the messages they have sent (and to whom) in the previous call to *SEND-PATH* or *CHECK*.

Algorithm 7 *INVESTIGATE* ▷ investigates the corruption situation

```
1: for each node, $q \notin \{\mathbf{s}, \mathbf{r}\}$, involved in the last call to SEND-PATH or CHECK do
2:     $q$ compiles all messages they have received (and from whom) and they have sent (and to whom) in the last call to SEND-PATH or CHECK.
3:     $q$ broadcasts these messages to all nodes in its quorum and the neighboring quorums.
4: end for
```

Algorithm 8 *MARK-IN-CONFLICTS* ▷ marks the nodes that are in conflict

```
1: for each pair of nodes, $(q_k, q_{k+1}) \in (Q_k, Q_{k+1})$, that is in conflict*, for $1 \leq k < \ell$ do
2:     node $q_{k+1}$ broadcasts a conflict message "$\{q_k, q_{k+1}\}$" to all nodes in $Q_{k+1}$,
3:     each node in $Q_v$ forwards "$\{q_k, q_{k+1}\}$" to all nodes in $Q_{k+1}$ and all nodes in $Q_k$,
4:     all nodes in $Q_k$ (or $Q_{k+1}$) send "$\{q_k, q_{k+1}\}$" to all other quorums that have node $q_k$ (or $q_{k+1}$).
5:     all nodes in each quorum that has $q_x$ or $q_{k+1}$ send "$\{q_k, q_{k+1}\}$" to the neighboring quorums.
6: end for
7: for each node, $q$, that receives a conflict message "$\{q_k, q_{k+1}\}$" do
8:     $q$ marks the nodes $q_k$ and $q_{k+1}$ in its marking table.
9: end for
10: if (0.49)-fraction of nodes in any quorum have been marked then
11:     each of these nodes is set unmarked in all quorums.
12:     each of these nodes is set unmarked in all its neighboring quorums.
13: end if
```

* A pair of nodes, (q_k, q_{k+1}) is *in conflict* if: 1) q_k was scheduled to send a message to q_{k+1} at some point in the last call to *SEND-PATH* or *CHECK*; and 2) q_{k+1} does not receive an expected message from q_k in *INVESTIGATE*, or q_{k+1} receives a message in *INVESTIGATE* that is different from the message that it has received from q_k in the last call to *SEND-PATH* or *CHECK*.

As a result of the investigation, *HEAL* identifies at least one pair of nodes that are in conflict. Note that a pair of nodes are in conflict if they each have broadcasted messages that are in conflict with the messages broadcasted by the other. Each pair of nodes that has been identified that they are in conflict, are marked in their quorums and the neighboring quorums. (See *MARK-IN-CONFLICTS* that is stated formally as Algorithm 8).

Moreover, each pair of nodes that are in conflict has at least one bad node. Thus, at most one good node is marked in each call to *HEAL*. In order to keep providing unmarked nodes to participate in *SEND-PATH* and *CHECK*, we set the constraint that if a $(1/2-\gamma)$-fraction of nodes in any quorum has been marked, they are set unmarked in all their quorums and the neighboring quorums.

Even though we unmark nodes in some situation, we provide a potential function argument (Lemma 3.7.12), which shows that all bad nodes are marked after $O(t)$ calls to *HEAL*. After all bad nodes are marked, no more corruptions occur.

3.7 Analysis

In this section, we provide the analysis of our algorithms. We first prove that *CHECK1* succeeds to detect corruptions with probability more than 1/2. Then, we prove the lemmas required for Theorem 3.3.1, in which *SEND* calls *CHECK2*. Note that we let all logarithms be base 2 throughout this section.

3.7.1 *CHECK1*

First, we show that if a message is corrupted in *SEND-PATH*, *CHECK1* succeeds to detect such a corruption with probability more than 1/2.

Lemma 3.7.1. *If $\ell \leq \frac{\log^2 n}{2}$, then CHECK1 fails to detect any message corruption with probability less than* 1/2*.*

Proof. *CHECK1* succeeds in detecting the message corruption if every subquorum has at least one good node.

Note that the fraction of bad nodes in any quorum is at most $1/4$. Note further that at least $(1/2+\gamma)$-fraction of the nodes in any quorum are unmarked, for $\gamma > 0$. Thus, the probability that an unmarked bad node is selected uniformly at random is at most $\frac{1/2}{1+2\gamma}$. Therefore, the probability that any subquorum of size $2\log\log n$ has only bad nodes is less than

$$(1/2)^{2\log\log n} = 1/\log^2 n.$$

Union-bounding over all ℓ subquorums, the probability that *CHECK1* fails to detect corruptions is less than $\ell/\log^2 n$. For $\ell \leq \frac{\log^2 n}{2}$, the probability that *CHECK1* fails is less than $1/2$. $\square$

3.7.2 *CHECK2*

In order to show that *CHECK2* succeeds to detect corruptions with probability at least $1/2$, we first define the *deception interval.*

Definition 3.7.2. *A deception interval, $d_i(j,k)$, is a path of unmarked bad nodes, x_{iw}'s, that are added to the subquorums, S_w's, in round i, for $1 < j \leq w \leq k < \ell$, such that: 1) Q_{j-1} is the rightmost quorum that has at least one good node, which provides the correct message to node x_{ij}; and 2) Q_{k+1} is the leftmost quorum that has at least one good node, to which node x_{ik} is scheduled to send and does not provide the correct message.*

Note that we say a deception interval, $d_i(j,k)$, in round i expands in any subsequent round if there exists a deception interval, $d_{i'}(j',k')$, in round $i' > i$ such that $j' < j \leq k'$ or $j' \leq k < k'$.

Note further that we say a deception interval, $d_i(j,k)$, in round i shrinks to length x in round $i' > i$ if there exists a deception interval, $d_{i'}(j', k')$, in round i' such that $j \leq j' \leq k' \leq k$ and $x = k' - j' + 1 < k - j + 1$.

Now we prove that 1) any deception interval never expands; and 2) any deception interval shrinks logarithmically from round to round. This will imply that in $O(\log^* n)$ rounds, any deception interval shrinks to length zero at which the corruption is detected.

Lemma 3.7.3. *Any deception interval in any round never expands in any subsequent round; otherwise, HEAL will be called.*

Proof. For each deception interval, $d_i(j, k)$, we have the following.

All good nodes in Q_{j-1}, that have been selected and have received k_p in rounds i or less, must receive uncorrupted messages signed by k_s, in all rounds subsequent to i; otherwise, *HEAL* will be called. Those good nodes, that receive the message signed by k_s, will send this message to 1) node $x_{ij} \in d_i(j, k)$; and 2) node $x_{i'j'} \in d_{i'}(j', k')$, for all $i' > i$, $j' \geq j$ and $k' \leq k$.

Moreover, all good nodes in Q_{k+1}, that have not received the correct message from node $x_{ik} \in d_i(j, k)$, must not receive this message from node $x_{i'k'} \in d_{i'}(j', k')$, for all $i' > i$, $j' \geq j$ and $k' \leq k$; otherwise, they will call *HEAL*. □

Now we prove that any deception interval shrinks logarithmically from round to round.

Lemma 3.7.4. *When a coin is flipped x times independently given that each coin is tail with probability at most $(1/2 - \epsilon)$, for any constant $\epsilon > 0$, then the probability of having any substring of tails of length at least $\max(1, 2\log x)$ is at most $1/2$.*

Proof. The probability of having a specific substring of tails, of length at least $2\log x$, is at most

$$\left(\frac{1}{2} - \epsilon\right)^{2\log x} < \left(\frac{1}{2}\right)^{2\log x} = \frac{1}{x^2}.$$

Union bounding over all possible substrings of length $2\log x$, then the probability of having a substring of tails, of length at least $2\log x$, is less than $x\frac{1}{x^2}$. Thus, for $x \geq 2$, $\frac{1}{x} \leq \frac{1}{2}$; and for $x = 1$, the probability of having a substring of length at least $\max(1, 2\log x)$ is trivially at most $(1/2 - \epsilon)$ for $\epsilon > 0$. □

Corollary 3.7.5. *When a coin is flipped x times independently given that each coin is tail with probability at most $(1/2 - \epsilon)$, for any constant $\epsilon > 0$, then the probability of having any substring of tails of length at least $\max(1, \lfloor x/2 \rfloor)$ is at most $1/2$.*

Now let $f(n) = 2\log n$, and let $f^{(i)}(n)$ be the function of applying function f, i times, over n. Also, we let $\log^{(i)}(n)$ be the function of applying logarithm i times over n.

Fact 3.7.6. *$\forall n > 4$ and $\forall i \geq 1$ such that $\log^{(i)}(n) \geq 2$,*

$$f^{(i)}(n) \leq 4\log^{(i)}(n).$$

Proof. We prove by induction over $i \geq 1$ that for $n > 4$ and $\log^{(i)}(n) \geq 2$,

$$f^{(i)}(n) \leq 4\log^{(i)}(n).$$

Base case: for $i = 1$, by definition,

$$f(n) = 2\log n \leq 4\log n.$$

Induction hypothesis: for $\log^{(j)}(n) \geq 2$,

$$\forall j < i, f^{(j)}(n) \leq 4\log^{(j)}(n).$$

Induction step: by definition,

$$f^{(i)}(n) = f(f^{(i-1)}(n)).$$

By induction hypothesis, for $\log^{(i-1)}(n) \geq 2$,

$$f^{(i-1)}(n) \leq 4\log^{(i-1)}(n).$$

Then, we have

$$f^{(i)}(n) \leq f(4\log^{(i-1)}(n)) = 2\log(4\log^{(i-1)}(n)),$$

or equivalently,

$$f^{(i)}(n) \leq 2(2+\log^{(i)}(n)) \leq 4\log^{(i)}(n),$$

for $\log^{(i)}(n) \geq 2$. $\square$

Now let $f^*(n)$ be the smallest value i such that $f^{(i)}(n) \leq 16$.

Fact 3.7.7. $\forall n > 4, f^*(n) \leq \log^* n - 2$.

Proof. Let $j = \log^* n - 2$. We know that $2 < \log^{(j)}(n) \leq 4$. By Fact 3.7.6, we have

$$f^{(j)}(n) \leq 4\log^{(j)}(n) \leq 16.$$

Thus, by definition, $f^*(n) \leq j = \log^* n - 2$. $\square$

Lemma 3.7.8. *Assume that any deception interval of length x shrinks to length $2\log x$ in a successful step. Then, for any deception interval of length $x' > 16$, after $\log^* x' - 2$ successful steps, it shrinks to a length of at most 16.*

Proof. Fact 3.7.7 proves this lemma. $\square$

The next lemma shows that the algorithm *CHECK2* catches corruptions with probability at least 1/2.

Lemma 3.7.9. *Assume some node selected uniformly at random in the last call to SEND-PATH has corrupted a message in a quorum path of length $\ell \leq n$. Then when CHECK2 is called, with probability at least 1/2, some node will call HEAL.*

Proof. By Lemma 3.7.3, any deception interval never expands over rounds. For shrinking deception intervals over rounds, we make use of Lemma 3.7.4 to shrink logarithmically any deception interval of length more than 16; otherwise, deception intervals shrink geometrically using Corollary 3.7.5.

Let X_i be an indicator random variable that is equal 1 if the deception interval in round i shrinks logarithmically in round $i+1$; and 0 otherwise. Recall that the longest deception interval has length of at most $\ell \leq n$. By Lemma 3.7.8, after having at most $\log^* n - 2$ of X_i's equal 1, the longest deception interval of length more than 16 shrinks to a deception interval of length at most 16.

Also let Y_j be an indicator random variable that is equal 1 if the deception interval of length $x \leq 16$ in round j shrinks geometrically to a deception interval of length at most $\lfloor x/2 \rfloor$ in round $j+1$; and 0 otherwise.

We require at most $\log^* n - 2$ rounds in which the X_i's equal 1 to shrink the longest deception interval, d, of length at most n to a deception interval, d', of length at most 16. Further, we require at most 5 rounds in which the Y_j's equal 1, to shrink d' to length 0.

By Lemma 3.7.4, for all $1 \leq i \leq 5(\log^* n - 2)$, for each event $\xi = (X_1 = x_1, ..., X_{i-1} = x_{i-1})$ where $x_1, \ldots, x_{i-1} \in \{0, 1\}$, $X_i \sim \mathit{Bernoulli}(p_i)$ for some $p_i \geq 1/2$. Similarly, by Corollary 3.7.5, for all $1 \leq j \leq 25$, for each event $\xi' = (Y_1 = y_1, ..., Y_{j-1} = y_{j-1})$ where $y_1, \ldots,$ $y_{j-1} \in \{0, 1\}$, $Y_j \sim \mathit{Bernoulli}(p_j)$ for some $p_j \geq 1/2$.

Let ξ_X be the event that $\sum_{i=1}^{5(\log^* n-2)} X_i \geq (\log^* n - 2)$, and let ξ_Y be the event that $\sum_{j=1}^{25} Y_j \geq 5$.

We know that the probability, that any deception interval of size n shrinks to size zero, is at least

$$\Pr(\xi_X \cap \xi_Y) = \Pr(\xi_Y | \xi_X) \cdot \Pr(\xi_X) = \Pr(\sum_{j=1}^{25} Y_j \geq 5) \cdot \Pr(\sum_{i=1}^{5(\log^* n-2)} X_i \geq (\log^* n - 2)).$$

For all $1 \leq k \leq 5(\log^* n + 3)$, let $Z_k \sim Bernoulli(1/2)$, and all Z_k's are independent random variables. With inequalities in terms of stochastic dominance [15, 36, 47, 78], for all i, j, k, $X_i \geq_{st} Z_k$ and $Y_j \geq_{st} Z_k$.

$$\Pr(\xi_Y|\xi_X) \cdot \Pr(\xi_X) \geq \Pr\left(\sum_{j=1}^{25} Z_j \geq 5\right) \cdot \Pr\left(\sum_{i=1}^{5(\log^* n-2)} Z_i \geq (\log^* n - 2)\right).$$

Note that $\mathbf{E}\left(\sum_{j=1}^{25} Z_j\right) = 25/2$, and $\mathbf{E}\left(\sum_{i=1}^{5(\log^* n-2)} Z_i\right) = 5(\log^* n - 2)/2$. Note further that for ξ_X, $n > 16$, or equivalently, $\log^* n \geq 4$.

By Chernoff bounds [53, 56], for $\delta = 3/5$, we have:

$$\Pr\left(\sum_{j=1}^{25} Z_j < 5\right) < e^{-\frac{9}{4}},$$

and

$$\Pr\left(\sum_{i=1}^{5(\log^* n-2)} Z_i < (\log^* n - 2)\right) < e^{-(5/4)(\log^* n-2)(3/5)^2} \leq e^{-\frac{9}{10}}.$$

This follows that $\Pr(\xi_Y|\xi_X) \cdot \Pr(\xi_X) \geq \left(1 - e^{-\frac{9}{4}}\right)\left(1 - e^{-\frac{9}{10}}\right) \geq 1/2$. Therefore, the probability that *CHECK2* succeeds in finding a corruption and calling *HEAL* is at least 1/2. □

Corollary 3.7.10. *If the number of rounds in CHECK2 is* $13(\log^* n + 3)$*, then corruptions are detected with probability at least* 0.99*.*

Moreover, Table 3.1 shows that the more rounds *CHECK2* has, the more probability of detecting corruptions is.

# Rounds	Probabilities
$6(\log^* n + 3)$	0.7
$7(\log^* n + 3)$	0.8
$8(\log^* n + 3)$	0.85
$9(\log^* n + 3)$	0.9

Table 3.1: # rounds versus probability of detecting corruptions

3.7.3 *HEAL*

Lemma 3.7.11. *If some node selected uniformly at random in the last call to SEND-PATH has corrupted a message, then the algorithm HEAL will identify a pair of neighboring quorums Q_j and Q_{j+1}, for some $1 \leq j < \ell$, such that at least one pair of nodes in these quorums is in conflict and at least one node in such pair is bad.*

Proof. First we show that if a pair of nodes x and y is in conflict, then at least one of them is bad. Assume not. Then both x and y are good. Then node x would have truthfully reported what it received; any message that x received would have been sent directly to y; and y would have truthfully reported what it received from x. But this is a contradiction, since for x and y to be in conflict, y must have reported that it received from x something different than what x reported receiving.

Now consider the case where a selected unmarked bad node corrupted a message in the last call to *SEND-PATH*. By the definition of corruption, there must be two good nodes q_j and q_k such that $j < k$ and q_j received the message m' sent by node $\mathbf{s}$, and q_k did not. We now show that some pair of nodes between q_j and q_k will be in conflict. Assume this is not the case. Then for all x, where $j \leq x < k$, nodes q_x and q_{x+1} are not in conflict. But then, since node q_j received the message m', and there are no pairs of nodes in conflict, it must be the case that the node q_k received the message m'. This is a contradiction. Thus, *HEAL* will find two nodes that are in conflict, and at least one of them will be bad.

Now we prove that at least one pair of nodes is found to be in conflict as a result of calling *HEAL*. Assume that no pair of nodes is in conflict. Then for every pair of nodes x and y, such that x was scheduled to send a message to y during any round i of *CHECK2*, x and y must have reported that they received the same message in round i. In particular, this implies via induction, that for every round i, for all j, where $1 \leq j \leq \ell$, all nodes in the sets S_j must have broadcasted that they received the message m' that was initially sent by node $\mathbf{s}$ in round i. But if this is the case, the node x that initially called *HEAL* would have received no inconsistent messages. This is a contradiction since in such a case, node x would have been unsuccessful in trying to initiate a call to *HEAL*. Thus, some pair of nodes must be found to be in conflict, and at least one of them is bad. □

The next lemma bounds the number of times that *HEAL* must be called before all bad nodes are marked.

Lemma 3.7.12. *HEAL is called $O(t)$ times before all bad nodes are marked.*

Proof. By Lemma 3.7.11, if a message is corrupted in the last call to *SEND-PATH* and is caught by *CHECK*, then *HEAL* is called. *HEAL* identifies at least one pair of nodes that are in conflict.

Let p be the probability of selecting an unmarked bad node uniformly at random. Recall that the fraction of bad nodes in any quorum is at most 1/4 and at any moment the fraction of unmarked nodes in any quorum is at least $(1/2 + \gamma)$ for $\gamma > 0$. Thus, we have

$$p \leq \frac{1}{2}\left(\frac{1}{1+2\gamma}\right).$$

Now let b be the number of bad nodes that are marked, and let g be the number of good nodes that are marked. Further, we let

$$f(b, g) = b - \left(\frac{p}{1-p}\right) g.$$

For each corruption caught, at least one bad node is marked. This implies that b increases by at least 1 and g increases by at most 1. Note that $\frac{p}{1-p} < 1$. Thus, $f(b, g)$ increases by at least $(1 - \frac{p}{1-p}) > 0$.

Moreover, when a $(1/2 - \gamma)$-fraction of nodes in any quorum Q of size $|Q|$ get unmarked for a constant $\gamma > 0$, b decreases by at most $p(1/2 - \gamma)|Q|$ and g decreases by at least $(1 - p)(1/2 - \gamma)|Q|$. This implies that $f(b, g)$ further increases by at least 0.

Thus, $f(b, g)$ is monotonically increasing by at least $(1 - \frac{p}{1-p}) > 0$ for each corruption caught. Note that when all bad nodes are marked,

$$f(b, g) \leq t.$$

Hence, all bad nodes are marked after at most $\left(\frac{1-p}{1-2p}\right) t$, or equivalently, at most $\left(1 + \frac{1}{2\gamma}\right) t/2$, calls to *HEAL*. □

3.7.4 Our Theorem

Now we prove Theorem 3.3.1. Note that we consider that *SEND* calls *CHECK2*.

Theorem 3.3.1 Assume we have a network with n nodes and $t \leq (1/4 - \epsilon)n$ bad nodes, for any constant $\epsilon > 0$. Then our algorithm has the following properties. 1) In an amortized sense, any call to *SEND* has $O(\ell + \log n)$ expected number of messages with $O(\ell)$ expected latency; and 2) the expected total number of times that *SEND* fails to deliver a message reliably is $O(t(\log^* n)^2)$.

Proof. We first show the message complexity and the latency of our algorithms. By Lemma 3.7.12, the number of calls to *HEAL* is $O(t)$. Thus, the resource cost of all calls to *HEAL* are bounded as the number of calls to *SEND* grows large. Therefore, for the amortized cost, we consider only the cost of the calls to *SEND-PATH* and *CHECK2*.

When sending a message through ℓ quorums, *SEND-PATH* has message cost $O(\ell + \log n)$ and latency $O(\ell)$. Recall that *CHECK2* has a message cost of $O((\ell + \log n)(\log^* n)^2)$ and a latency of $O(\ell \log^* n)$, but *CHECK2* is called only with probability $1/(\log^* n)^2$. Hence, the call to *SEND* has amortized expected message cost $O(\ell + \log n)$ and amortized expected latency $O(\ell)$.

More specifically, if we perform any number of message sends through quorum paths, where ℓ_M is the longest such path, and $\mathcal{L}$ is the sum of the quorums traversed in all such paths, then the expected total number of messages sent will be $O(\mathcal{L}+t\cdot(\ell_M \log^2 n+\log^5 n))$, and the latency is $O(t\cdot\ell_M)$.

This is true since each call to *HEAL* has message cost $O(\ell_M \log^2 n + \log^5 n)$ and latency $O(\ell_M)$, where:

1. the node, x, making the call to *HEAL* broadcasts its reason of calling *HEAL* to all nodes in its quorum, this has message cost $O(\log n)$ and latency $O(1)$;
2. all nodes in every quorum in the quorum path are notified via all-to-all communication when *HEAL* is called, these notifications have a message cost of $O(\ell_M \log^2 n)$ and a latency of $O(\ell_M)$;
3. *HEAL* has $O(\log^* n)$ broadcasts over at most ℓ_M quorums, that has message cost $O(\ell_M \log^* n \cdot \log n)$ and latency $O(1)$;
4. the message cost when all nodes in Q_1 and Q_ℓ broadcast is $O(\log^2 n)$ with latency $O(1)$;
5. marking a pair of nodes that are in conflict has message cost $O(\log^3 n)$ and latency $O(1)$; and
6. note that marking a pair of nodes that are in conflict could cause $O(\log n)$ quorums to be unmarked. Note further that unmarking $O(\log n)$ nodes in any quorum has a message cost of $O(\log^4 n)$. Thus, unmarking $O(\log n)$ quorums has message cost $O(\log^5 n)$ and latency $O(1)$.

Now we show the expected total number of corruptions. Recall that by Lemma 3.7.12, the number of calls to *HEAL* before all bad nodes are marked is $O(t)$. Thus, *CHECK2* must detect corruptions and calls *HEAL* $O(t)$ times. Moreover, if a bad node caused a corruption during a call to *SEND-PATH*, then by Lemmas 3.7.9 and 3.7.11, with probability at least 1/2, *CHECK2* will catch it. Note that *CHECK2* is called with probability $\frac{1}{(\log^* n)^2}$. Therefore, the expected total number of corruptions is $O(t(\log^* n)^2)$. □

3.8 Empirical Results

3.8.1 Setup

In this section, we empirically compare between two algorithms in terms of the message cost, the latency, the fraction of messages corrupted and the expected total number of corruptions, via simulation.

The first algorithm we simulate is *no-self-healing* algorithm from [29]. This algorithm has no self-healing properties, and simply uses all-to-all communication between quorums that are connected in a butterfly network. The second algorithm is *self-healing*, wherein we apply our self-healing algorithm in the butterfly networks triggering *CHECK1* and *CHECK2* separately.

In our experiments, we consider butterfly networks of sizes up to $n = 30{,}509$, where $\ell = \lfloor \log n \rfloor - 2$ and the quorum size is $\lfloor 4 \log n \rfloor$.

In one experiment, *SEND* calls *CHECK1* with probability $1/(\log\log n)^2$ and with subquorum size $\lfloor 2 \log\log n \rfloor$. Another experiment has *SEND* trigger *CHECK2* with probability $1/(\log^* n)^2$ and with subquorum size $\lfloor 2 \log^* n \rfloor$.

Moreover, we do our experiments for several fractions of bad nodes such as f equal to 1/8, 1/16, 1/32 and 1/64, where $f = t/n$. Note that for larger f, marking and unmarking processes are performed more frequently. This makes the simulation take longer to eventually mark all bad nodes.

Our simulations consist of a sequence of calls to *SEND* over the network, given a pair of nodes $\mathbf{s}, \mathbf{r}$, chosen uniformly at random, where node $\mathbf{s}$ sends a message to node $\mathbf{r}$. We simulate an adversary who at the beginning of each simulation chooses uniformly at random without replacement a fixed number of nodes to control. Our adversary attempts to corrupt messages between nodes whenever possible. Aside from attempting to corrupt messages, the adversary performs no other attacks.

3.8.2 Results

The results of our experiments are shown in Figures 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17, 3.18 and 3.19. Our results highlight two strengths of our self-healing algorithms (*self-healing*) when compared to algorithms without self-healing (*no-self-healing*). First, the message cost per call to *SEND* decreases as the total number of calls to *SEND* increases. Second, for a fixed number of calls to *SEND*, the message cost per call to *SEND* decreases as the total number of bad nodes decreases. In particular, when there are no bad nodes, *self-healing* has dramatically less message cost than *no-self-healing*.

In our experiments, we show the following for n = 14,116 and n = 30,509: 1) the expected number of messages per call to *SEND*; 2) the expected latency per call to *SEND*; 3) the fraction of messages corrupted for each call to *SEND*; and 4) the expected total number of corruptions. Moreover, we show the improvement factor of # messages that is the ratio of the expected number of messages of the *no-self-healing* algorithm to the expected number of messages of *SEND*; and we show the latency increase factor that is the ratio of the expected latency of *SEND* to the expected latency of *no-self-healing* algorithm.

Expected Number of Messages

Figures 3.10 and 3.11 show that the expected number of messages per call to *SEND* decreases as the total number of calls to *SEND* increases, and for a fixed number of calls to *SEND*, the expected number of messages per call to *SEND* decreases as f decreases.

Table 3.2 shows that when all bad nodes are marked, *self-healing* has dramatically less expected number of messages per call to *SEND* than *no-self-healing*.

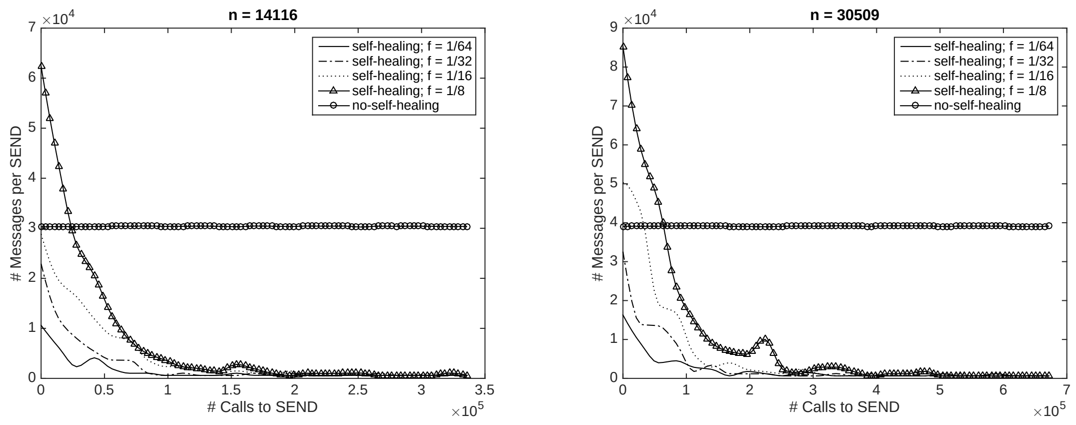

Figure 3.10: # messages per call to *SEND* versus # calls to *SEND*, for n = 14,116 and n = 30,509, when *SEND* calls *CHECK1*.

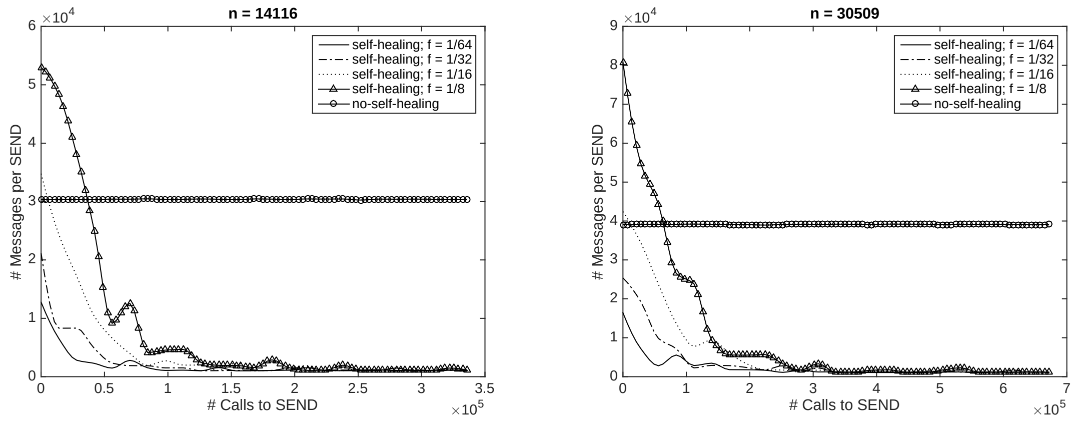

Figure 3.11: # messages per call to *SEND* versus # calls to *SEND*, for n = 14,116 and n = 30,509, when *SEND* calls *CHECK2*.

n	*self-healing*		*no-self-healing*
	CHECK1	*CHECK2*	
14,116	598	1,078	30,390
30,509	649	1,177	39,100

Table 3.2: Expected # messages per call to *SEND* in *self-healing* and in *no-self-healing* for n = 14,116 and n = 30,509.

Improvement Factor of # Messages

Figures 3.12 and 3.13 show the ratio of the expected number of messages of the *no-self-healing* algorithm to the expected number of messages of *SEND* (for *CHECK1* and *CHECK2*), after all bad nodes are marked. They show that the expected number of messages has been improved by a factor of $O(\log^2 n)$.

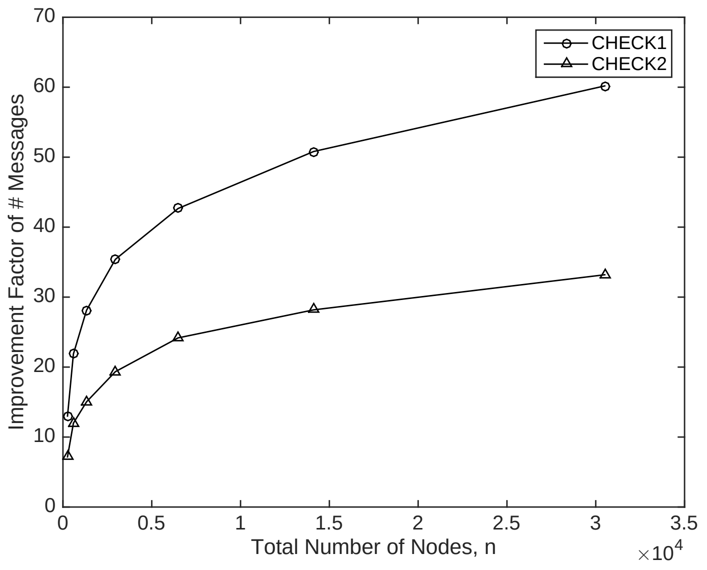

Figure 3.12: Improvement Factor of # Messages for *CHECK1* and *CHECK2*.

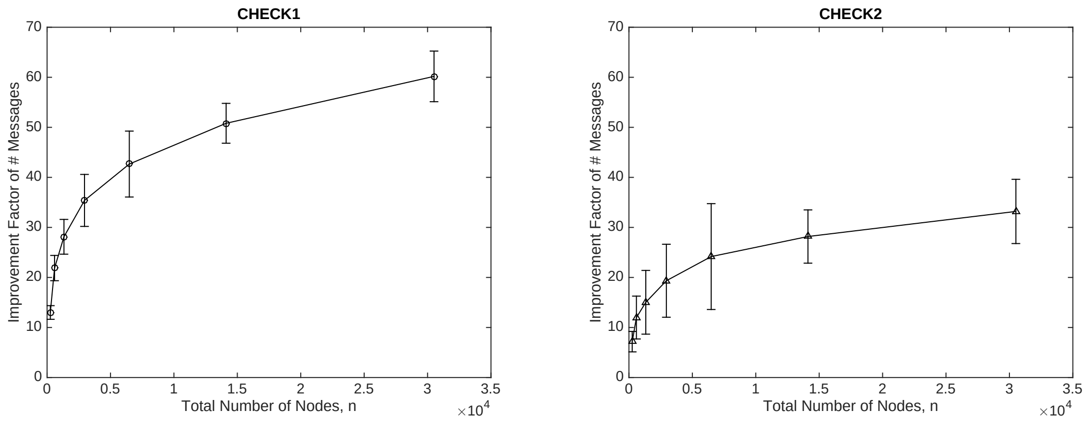

Figure 3.13: Improvement Factor of # Messages for *CHECK1* and *CHECK2* with Error Bars

Expected Latency

Figures 3.14 and 3.15 show that the latency of *no-self-healing* is less than the latency of *self-healing* due to the latency of *CHECK1* and *CHECK2*. Note that the unit of latency is message hop.

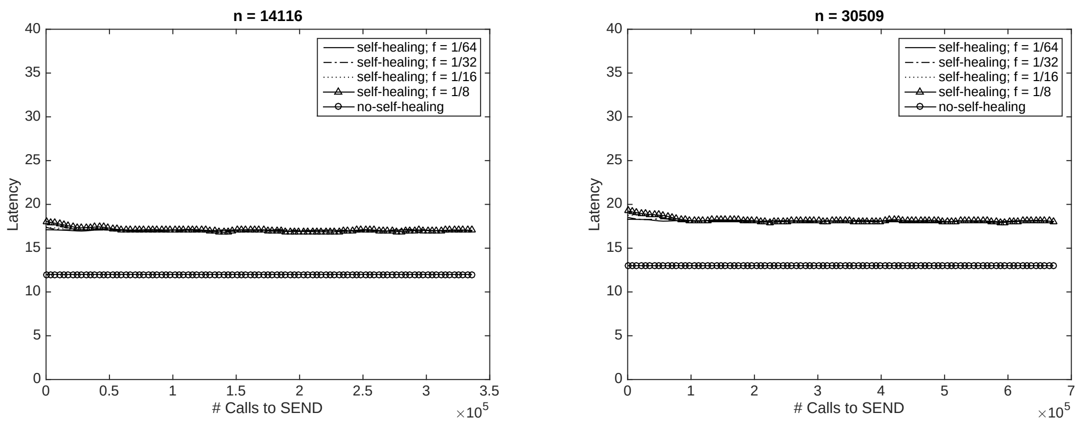

Figure 3.14: Latency per call to *SEND* versus # calls to *SEND*, for n = 14,116 and n = 30,509, when *SEND* calls *CHECK1*.

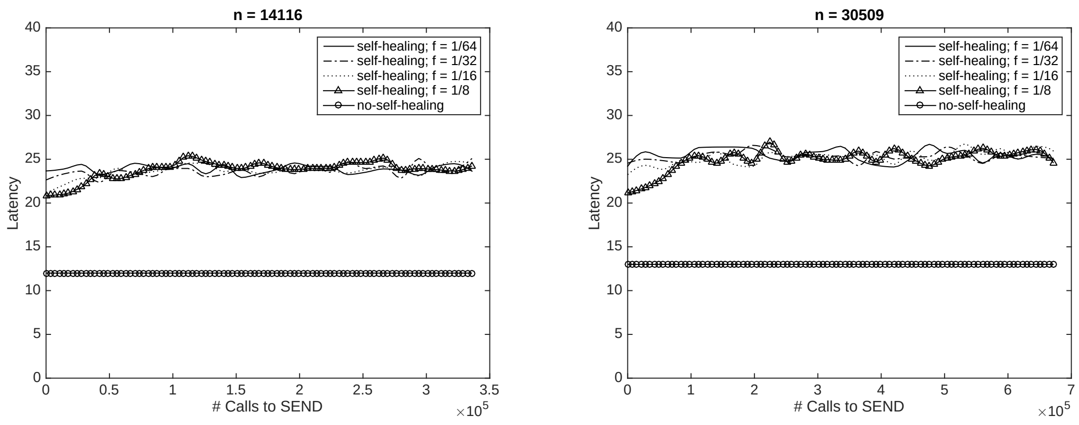

Figure 3.15: Latency per call to *SEND* versus # calls to *SEND*, for n = 14,116 and n = 30,509, when *SEND* calls *CHECK2*.

Table 3.3 shows that for n = 14,116 and n = 30,506, after all bad nodes are marked, we have that: 1) *self-healing* calling *CHECK2* has more latency than *self-healing* calling *CHECK1*; and 2) the latency of *self-healing* is at most twofold the latency of *no-self-healing*.

n	*self-healing*		*no-self-healing*
	CHECK1	*CHECK2*	
14,116	17	23	12
30,509	18	25	13

Table 3.3: Expected latency per call to *SEND* in *self-healing* and in *no-self-healing* for $n = 14{,}116$ and $n = 30{,}509$.

Latency Increase Factor

Figures 3.16 and 3.17 show the ratio of the expected latency of *SEND* (for *CHECK1* and *CHECK2*) to the expected latency of the *no-self-healing* algorithm, after all bad nodes are marked or all selected leaders are good. They show that the expected latency increases by a factor of $\Theta(1)$.

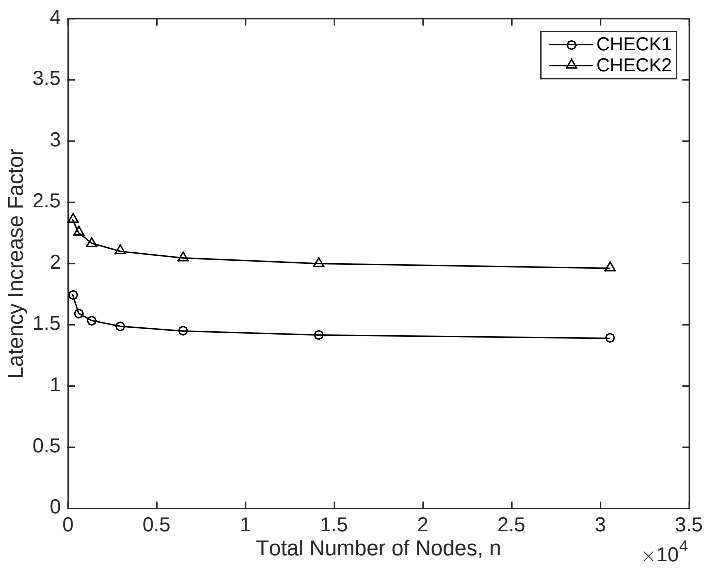

Figure 3.16: Latency Increase Factor for *CHECK1* and *CHECK2*

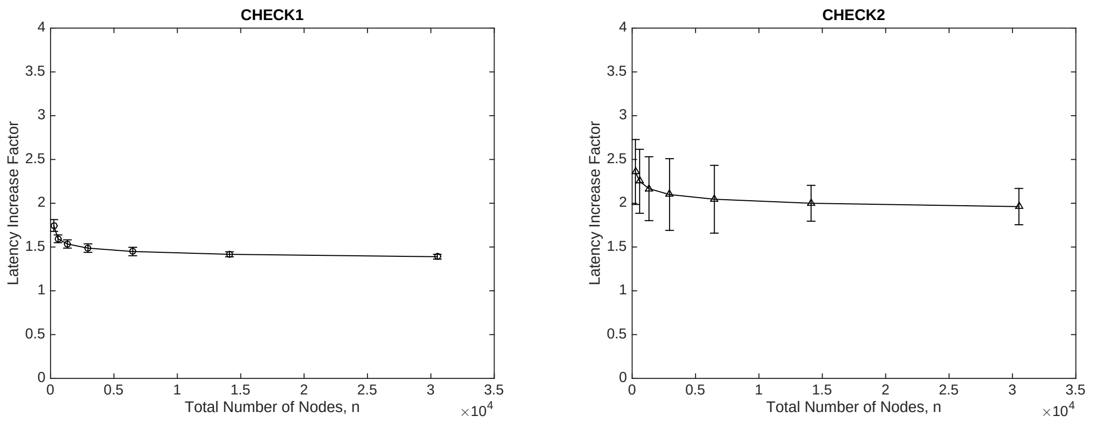

Figure 3.17: Latency Increase Factor for *CHECK1* and *CHECK2* with Error Bars

Fraction of Messages Corrupted

The fraction of messages corrupted per call to *SEND* shows the probability that the message is corrupted in this call.

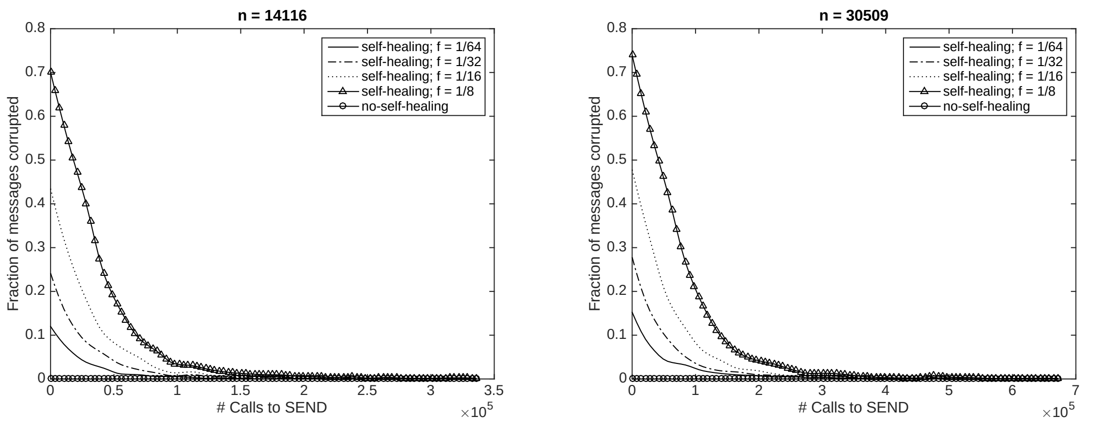

Figure 3.18: Fraction of messages corrupted versus # calls to *SEND*, n = 14,116 and n = 30,509, when *SEND* calls *CHECK1*.

In Figures 3.18 and 3.19, *no-self-healing* has no corruptions; however, for *self-healing*, the fraction of messages corrupted per call to *SEND* decreases as the total number of calls to *SEND* increases. Also, for a fixed number of calls to *SEND*, the fraction of messages corrupted per call to *SEND* decreases as the total number of bad nodes decreases.

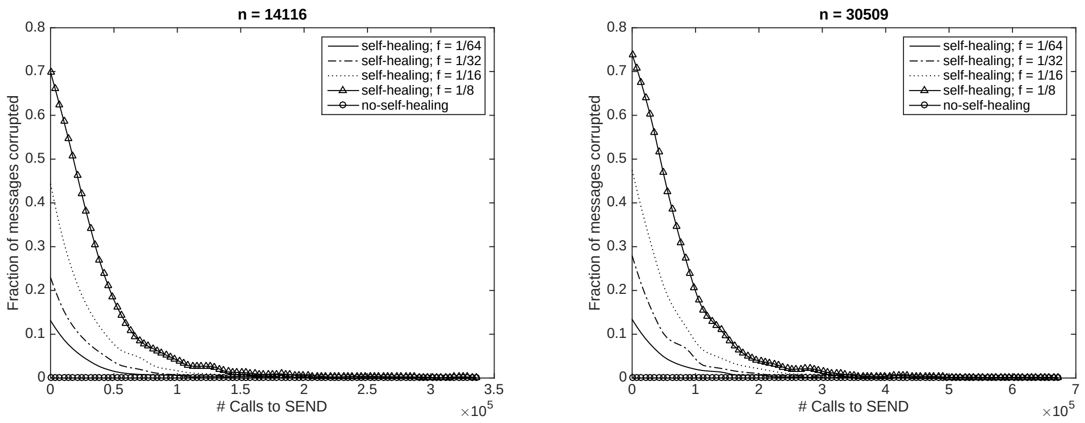

Figure 3.19: Fraction of messages corrupted versus # calls to *SEND*, for n = 14,116 and n = 30,509, when *SEND* calls *CHECK2*.

Expected Total Number of Corruptions

In Figures 3.18 and 3.19, for each network given the number of nodes and the fraction of bad nodes, if we integrate the corresponding curve, then we get the total number of times that the message is corrupted in all calls to *SEND* in this network.

Tables 3.4, 3.5, 3.6 and 3.7 show the fact that when *SEND* calls *CHECK1*, the expected total number of corruptions is at most Σ_1, where

$$\Sigma_1 = 2\left(\frac{1-2f}{1-4f}\right)t(\log^* n)^2,$$

and when *SEND* calls *CHECK2*, the expected total number of corruptions is at most Σ_2, where

$$\Sigma_2 = 2\left(\frac{1-2f}{1-4f}\right)t(\log\log n)^2.$$

f	total # corruptions	Σ_1
1/64	3,457	4,102
1/32	6,930	8,507
1/16	13,831	18,526
1/8	27,721	47,641

Table 3.4: Total # corruptions when *SEND* calls *CHECK1* for $n =$ 14,116.

f	total # corruptions	Σ_2
1/64	3,454	7,293
1/32	6,918	15,124
1/16	13,845	32,859
1/8	27,685	84,696

Table 3.5: Total # corruptions when *SEND* calls *CHECK2* for $n =$ 14,116.

f	total # corruptions	Σ_1
1/64	7,490	8,866
1/32	14,996	18,386
1/16	29,949	40,042
1/8	59,932	102,967

Table 3.6: Total # corruptions when *SEND* calls *CHECK1* for $n =$ 30,509.

f	total # corruptions	Σ_2
1/64	7,498	15,762
1/32	14,989	32,687
1/16	29,970	71,187
1/8	59,969	183,054

Table 3.7: Total # corruptions when *SEND* calls *CHECK2* for $n =$ 30,509.

Chapter 4

Self-Healing Computation

"Computations are everywhere, once you begin to look at things in a certain way."

– Rudy Rucker

4.1 Introduction

How can we protect a network against adversarial attack? A traditional approach provides robustness through redundant components. If one component is attacked, the remaining components maintain functionality. Unfortunately, this approach incurs significant resource cost, even when the network is not under attack.

An alternative approach is self-healing, where a network detects the damage made by attacks, inspects the corruption situation and automatically recovers. Self-healing algorithms expend additional resources only when it is necessary to repair from attacks.

In this chapter, we describe self-healing algorithms for the problem of *reliable multiparty computation (RMC)*. In the RMC problem, there are n parties, each with an individual input, and the parties want to jointly compute a function f over n inputs. A hidden 1/4-fraction of the parties are controlled by an omniscient Byzantine adversary. A party that is controlled by the adversary is said to be *bad*, and the remaining parties are said to be *good*. Our goal is to ensure that all good parties learn the output of f. [1]

RMC abstracts many problems that may occur in high-performance computing, sensor networks, and peer-to-peer networks. For example, we can use RMC to enable performance profiling and system monitoring, compute order statistics, and enable public voting.

Our main result is an algorithm for RMC that 1) is asymptotically optimal in terms of total messages and total computational operations; and 2) limits the expected total number of corruptions. Ideally, each bad party would cause $O(1)$ corruptions; in our algorithm, each bad party causes $O((\log^* m)^2)$ corruptions in expectation.

This chapter is organized as follows. In Section 4.2, we describe our model. Our main theorem is given in Section 4.3, and we provide a technical overview in Section 4.4. The related work is discussed in Section 4.5. Section 4.6 describes our algorithms. The analysis of our algorithms is shown in Section 4.7. Section 4.8 gives empirical results showing how our algorithms improve the efficiency of the butterfly networks of [30].

4.2 Our Model

We assume a *static* Byzantine adversary that takes over $t \leq (\frac{1}{4} - \epsilon)n$ parties before the algorithm begins, for any constant $\epsilon > 0$. As mentioned previously, parties that are compromised by the adversary are called *bad*, and the remaining parties are *good*. The bad parties may arbitrarily deviate from the protocol, by sending no messages, excessive numbers of messages, incorrect messages, or any combination of these. The good parties follow the protocol. We assume that the adversary knows our protocol, but is unaware of the random bits of the good nodes. We make use of a public key cryptography scheme, and thus we assume that

[1] Note that RMC differs from secure multiparty computation (MPC) only in that there is no requirement to keep inputs private.

the adversary is computationally bounded.

Also, we assume a partially synchronous communication model, where any message sent from one good node to another good node requires at most h time steps to be sent and received, and the value h is known to all nodes. We allow the adversary to be *rushing* in the sense that the bad nodes receive all messages from good nodes in a round before sending out their own messages.

We further assume that each party has a unique ID. We say that party p has a link to party q if p knows q's ID and can thus directly communicate with node q.

In the reliable multiparty computation problem, we assume that the function f can be implemented with an arithmetic circuit over m gates, where each gate has two inputs and at most two outputs.[2] For simplicity of presentation, we focus on computing a single function multiple times (with changing inputs). However, we can also compute multiple functions with our algorithm.

4.3 Our Result

We describe an algorithm, *COMPUTE*, to efficiently solve reliable multiparty computation. Our main result is summarized in the following theorem.

Theorem 4.3.1. *Assume we have n parties providing inputs to a function f that can be computed by an arithmetic circuit with depth ℓ and containing m gates. Then COMPUTE solves RMC and has the following properties.*

(1) In an amortized sense[3], any execution of COMPUTE requires $O(m+n\log n)$ messages

[2]We note that any gate of any fixed in-degree and out-degree can be converted into a fixed number of gates with in-degree 2 and out-degree at most 2.

[3]In particular, if we call *COMPUTE* $\mathcal{L}$ times, then the expected total number of messages sent will be $O(\mathcal{L}(m+n\log n)+t(m\log^2 n))$. Since t is fixed, for large $\mathcal{L}$, the expected number of messages per *COMPUTE* is $O(m+n\log n)$. Similar for the cost of computational operations.

sent by all parties, $O(m + n \log n)$ computational operations performed by all parties, and $O(\ell)$ latency.

(2) The expected total number of times that COMPUTE returns a corrupted output is $O(t(\log^* m)^2)$.

The theoretical result of this chapter was first presented as an extended abstract in [72].

Our experimental results (Section 4.8) show that our algorithms reduce the message cost, compared to the naive algorithm, by a factor of 65 for $n = 8{,}191$.

4.4 Technical Overview

Our algorithms make critical use of quorums and a quorum graph.

4.4.1 Quorum Graph of Computation Network

We define a quorum to be a set of $\Theta(\log n)$ parties, of which at most 1/4-fraction are bad. Many results show how to create and maintain a network of quorums [14, 30, 31, 39, 46, 59, 60, 76]. All of these results maintain what we will call a *quorum graph* in which each vertex represents a quorum. The properties of the quorum graph are:

(1) each party is in $\Theta(\log n)$ quorums;

(2) for any quorum Q, any party in Q can communicate directly to any other party in Q; and

(3) for any quorums Q and Q' that are connected in the quorum graph, any party in Q can communicate directly with any party in Q' and vice versa.

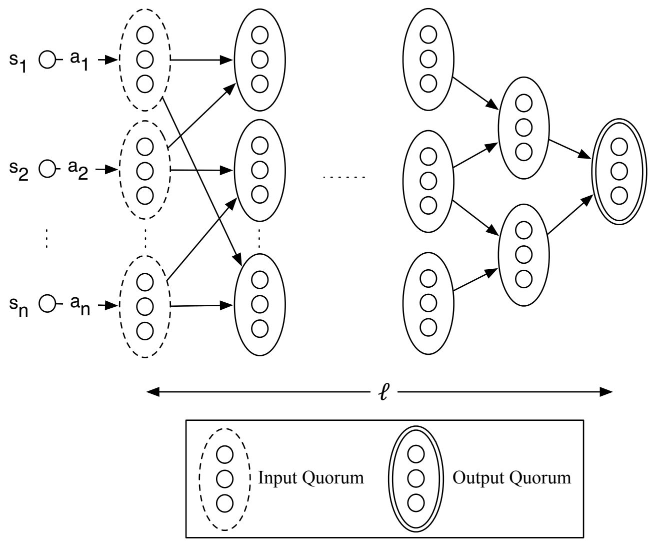

Figure 4.1: Quorum Graph in Computation Network

Moreover, we assume that for any two parties x and y in a quorum, x knows all quorums that y is in.

4.4.2 Computing with Quorums

We maintain a quorum graph with $m + n$ nodes: m nodes for the gates of the circuit and n nodes for the inputs of the parties. The input nodes are connected to the gates using these inputs, and the gate nodes are connected as in the circuit. Quorums are mapped to nodes in this quorum graph as described above.

Figure 4.1 shows the quorum graph in which the computation is performed from the left to the right. In particular, the input quorums are the leftmost quorums and the output quorum is the rightmost quorum in the quorum graph.

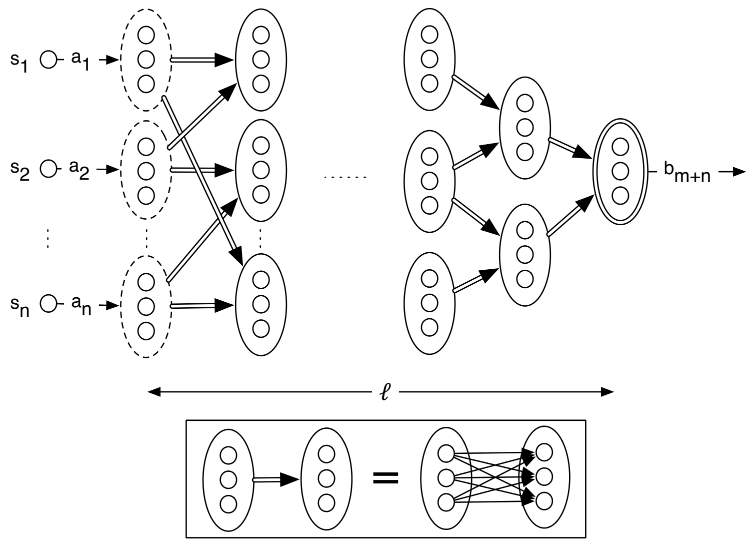

Figure 4.2: Computing through a circuit of quorums via all-to-all communication

4.4.3 Naive Computation

A correct but inefficient way to solve RMC is as follows. Each party s_i sends its input to all parties of the appropriate input quorum. Then the computation is performed from left to right. All parties in each quorum compute the appropriate gate operation on their inputs, and send their outputs to all parties in the right neighboring quorums via all-to-all communication. At the next level, all parties in each quorum take the majority of the received messages in order to determine the correct input for their gate. At the end, the parties in the rightmost quorum will compute the correct output of the circuit (See Figure 4.2). They then forward this output back from right to left through the quorum graph using the same all-to-all communication and majority filtering (See Figure 4.3).

Unfortunately, this naive algorithm requires $O((m+n)\log^2 n)$ messages and $O(m \log n)$ computational operations. Our main goal is to remove polylog and log factors. [4]

[4]We note that such asymptotic improvements can be significant for large networks. For example, if $n = 4k$, then our algorithm reduces message cost by a factor of $O(\log^2 n) = 336$.

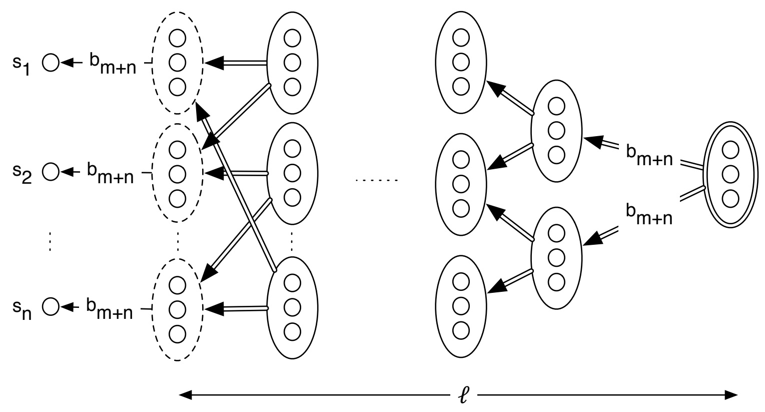

Figure 4.3: Sending back the result to all senders through a circuit of quorums via all-to-all-communication

4.4.4 Our Approach

A more efficient approach is for each quorum to have a leader, and for this leader to receive inputs, perform gate computations, and send off the output. Unfortunately, a single bad leader can corrupt the entire computation.

To address this issue, we provide *CHECK* (Section 4.6.3). This algorithm determines if there has been a corruption, and if so, it calls *RECOVER* (Section 4.6.4), which identifies at least one pair of parties that are in *conflict*. Informally, we say that a pair of parties are in conflict if they each accuse the other of malicious behavior. In such a situation, we know that at least one party in the pair is bad. Our approach is to mark both parties in each conflicting pair, and these marked parties are prohibited from participating in future computation but they still can provide the inputs of the circuit. [5]

The basic idea of *CHECK* is to redo the computation through subsets of parties; one subset for each gate. *CHECK* runs in multiple rounds. Initially, all subsets are empty; and

[5]A technical point is that we may need to unmark all parties in a quorum if too many parties in that quorum become marked. However, a potential function argument (Lemma 4.7.17) shows that after $O(t)$ markings, all bad parties will be marked.

in each round, a new party is selected uniformly at random from each quorum to be added to each subset. We call these parties the *checkers*. For convenience of presentation, we will refer to the leaders as the checkers for round 0. For each round $i \geq 1$, all i checkers at gate g: 1) receive inputs to g from the checkers at each input gate for g; 2) compute the gate output for g based on these inputs; and 3) send this output to the checkers at each output gate for g. If a good checker ever receives inconsistent inputs, it calls *RECOVER*. Unfortunately, waiting until a round where each gate has had at least one good checker would require $O(\log n)$ rounds.

To do better, we use the following approach. Let G be the quorum graph as defined above and let the checkers be selected as above. Call a subgraph of G bad in a given round if all checkers in the nodes of that subgraph are bad; note that such a subgraph consists of the new checkers that are added to the subsets in that round. When the adversary corrupts an output of a bad subgraph of G in one round, it has to keep corrupting this output by nesting levels of bad subgraphs of G in all subsequent rounds.

Recall that in each round, new checkers are selected uniformly at random. When *CHECK* selects a good checker at a quorum, it is as removing the node associated with this quorum from the quorum graph. Thus, we can view *CHECK* as repeatedly removing nodes from increasingly smaller subgraphs of G until no nodes remain, at which the corruption is detected. A key lemma (Lemma 4.7.4) shows that for any rooted directed acyclic graph (DAG), with m nodes and maximum indegree 2, when each node is deleted independently with probability at least $1/2 + \epsilon$, for any constant $\epsilon > 0$, the probability of having a connected DAG, rooted at one node, with surviving nodes of size $\Omega(\log m)$, is at most 1/2. By this lemma, we show that *CHECK* requires only $O(\log^* m)$ rounds to detect a corruption with constant probability.[6]

CHECK requires $O((m + n \log n)(\log^* m)^2)$ messages. Then, we can call it with proba-

[6]This probability can be made arbitrarily close to 1 by adjusting the hidden constant in the $O(\log^* m)$ rounds. (See Corollary 4.7.13)

bility $1/(\log^* m)^2$ and obtain asymptotically optimal resource costs for the RMC problem, while incurring $O(t(\log^* m)^2)$ corruptions in expectation.

4.5 Related Work

Our results are inspired by recent work on self-healing algorithms. Early work of [35, 40, 57, 81, 82] discusses different restoration mechanisms to preserve network performance by adding capacity and rerouting traffic streams in the presence of node or link failures. This work presents mathematical models to determine global optimal restoration paths, and provides methods for capacity optimization of path-restorable networks.

More recent work [19, 37, 38, 64, 73, 75] considers models where the following process repeats indefinitely: an adversary deletes some nodes in the network, and the algorithm adds edges. The algorithm is constrained to never increase the degree of any node by more than a logarithmic factor from its original degree. In this model, researchers have presented algorithms that ensure the following properties: the network stays connected and the diameter does not increase by much [19, 37, 73]; the shortest path between any pair of nodes does not increase by much [38]; expansion properties of the network are approximately preserved [64]; and keeping network backbones densely connected [75].

This paper particularly builds on [48]. That paper describes self-healing algorithms that provide reliable communication, with a minimum of corruptions, even when a Byzantine adversary can take over a constant fraction of the nodes in a network. While our attack model is similar to [48], reliable *computation* is more challenging than reliable communication, and hence this paper requires a significantly different technical approach. Additionally, we improve the fraction of bad parties that can be tolerated from 1/8 to 1/4.

Reliable multiparty computation (RMC) is closely related to the problem of secure multiparty computation (MPC) which has been studied extensively for several decades (see e.g. [6, 16, 17, 68, 83] or the recent book [67]). RMC is simpler than MPC in that it does not require inputs of the parties to remain private. Our algorithm for RMC is significantly more efficient than current algorithms for MPC, which require at least polylogarithmic blowup in communication and computational costs in order to tolerate a Byzantine adversary. We reduce these costs through our self-healing approach, which expends additional resources only when corruptions occur, and is able to "quarantine" bad parties after $O(t(\log^* m)^2)$ corruptions.

4.6 Our Algorithms

In this section, we describe our algorithms: *COMPUTE*, *EVALUATE*, *CHECK* and *RECOVER*. Our algorithms aim at detecting corruptions and marking the bad parties. Note that the parties that are marked are not allowed to participate in the computation; but they still can provide inputs to the circuit. Note further that all parties are initially unmarked.

Recall that there are n parties, each provides an input to an input quorum, Q_i, for $1 \leq i \leq n$; and then the computation is performed through m quorums, Q_j's, for $n+1 \leq j \leq m+n$. The result is produced at an output quorum Q_{m+n}, and it is sent back to the senders through the m quorums.

Before discussing our main *COMPUTE* algorithm, we describe that when a party x broadcasts a message msg, signed by the private key of a quorum Q, to a set of parties S, it calls $BROADCAST(msg, Q, S)$.

4.6.1 *BROADCAST*

In *BROADCAST* (Algorithm 9), we use threshold cryptography to avoid the overhead of Byzantine Agreement. In a (η, η')-threshold cryptographic scheme, a private key is distributed among η parties in such a way that 1) any subset of more than η' parties can jointly reassemble the key; and 2) no subset of at most η' parties can recover the key. The private key can be distributed using a *Distributed Key Generation* (DKG) protocol [45].

Algorithm 9 *BROADCAST* (msg, Q, S) ▷ A party x sends message msg to a set of parties S after signing it by the private key of quorum Q.

```
1: Party $x$ calls SIGN $(msg, Q)$.            ▷ signs $msg$ by the private key of quorum $Q$.
2: Party $x$ sends this signed-message to all parties in $S$.
```

In particular, we use ($|Q|$, $\frac{3|Q|}{4} - 1$)-DKG to generate for each quorum Q the following: 1) a (distributed) private key of Q, where a private key share is generated for each party in Q; 2) a public key of Q to verify each message signed by the (distributed) private key of Q; and 3) a public key share for each party in Q in order to verify any message signed by the private key share of this party.

Note that for each quorum, Q, the public key of Q and the public key share of each party in Q are known to all parties in Q and all parties in the neighboring quorums.

Recall that a party x calls *BROADCAST*(msg, Q, S) in order to send a message msg to all parties in S after signing msg by the private key of quorum Q. Signing a message msg, by the private key of Q, is formally stated in *SIGN* (msg, Q) (Algorithm 10). Note that we let the message msg be signed by the private key of Q in order to fulfill the following: 1) at least 3/4-fraction of the parties in quorum Q have received the same message msg; 2) they agree upon the content of msg; and 3) they give permission to x to broadcast this message.

Algorithm 10 *SIGN* (msg, Q) ▷ Signs message msg by the private key of quorum Q.

```
1: Party $x$ sends message $msg$ to all parties in $Q$.
2: Each party in $Q$ signs $msg$ by its private key share to obtain its message share.
3: Each party in $Q$ sends its message share back to party $x$.
4: Party $x$ interpolates at least $\frac{3|Q|}{4}$ message shares to obtain a signed-message of $Q$.
```

Any call to *BROADCAST* has $O(\log n + |S|)$ messages and $O(\log n)$ computational operations for signing the message msg by $O(\log n)$ parties in Q, with latency $O(1)$.

4.6.2 *COMPUTE*

Now we describe our main algorithm, *COMPUTE* (Algorithm 11), which calls *EVALUATE* (Algorithm 12).

```
Algorithm 11 COMPUTE     ▷ performs a reliable computation sending result to parties
1: EVALUATE              ▷ computes and sends back the result through a circuit of leaders.
2: TRIGGER-CHECK         ▷ Output quorum triggers CHECK with probability $1/(\log^* m)^2$.
```

In *EVALUATE*, the n parties broadcast their inputs to the input quorums; note that we assume that all parties provide their inputs to the circuit in the same round. The input quorums forward these inputs to a circuit of m leaders in order to perform the computation and provide the result to the output quorum (See Figure 4.4). Then this result is sent back to all senders (all parties) through the same circuit (See Figure 4.5). Note that we define a leader of a quorum as a representative party of all parties in this quorum, and its leadership is known to all parties in this quorum and the neighboring quorums.

In the presence of an adversary, *EVALUATE* is vulnerable to corruptions. Thus, *COMPUTE* calls *TRIGGER-CHECK* (Algorithm 13), in which the parties of the output quorum decide together, to trigger *CHECK* (Algorithm 14) with probability $1/(\log^* m)^2$, using secure multiparty computation (MPC) [6, 17, 68]. *CHECK* is triggered in order to detect with probability at least 1/2 if a computation was corrupted in the last call to *EVALUATE*.

Unfortunately, while *CHECK* can determine if a corruption occurred, it does not locate where the corruption originally occurred. Thus, when *CHECK* detects a corruption, *RECOVER* (Algorithm 19) is called. In each call to *RECOVER*, two neighboring quorums in the circuit are identified such that at least one pair of parties in these quorums is in conflict and at least one party in this pair is bad. Then the parties that are in conflict are marked in all quorums they are in, and in their neighboring quorums. Moreover, for each pair of leaders that are in conflict, their quorums elect a new pair of unmarked leaders uniformly at random. Note that if $(1/2-\gamma)$-fraction of parties in any quorum have been marked, for any constant $\gamma > 0$, e.g., $\gamma = 0.01$, they are set unmarked in all their quorums and in all their neighboring quorums.

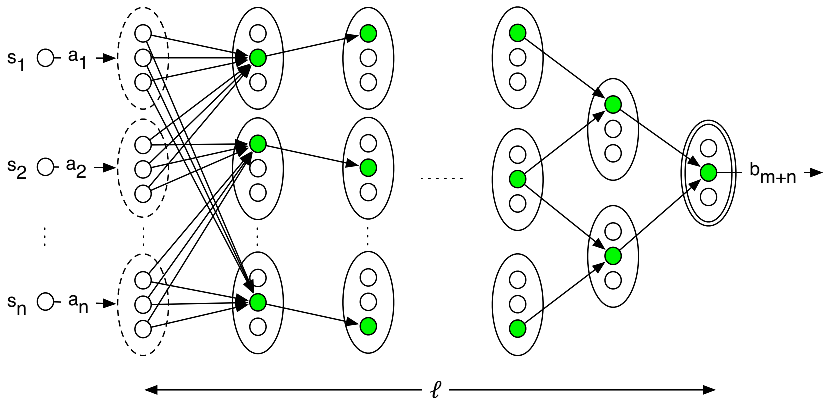

Figure 4.4: Computing through a circuit of leaders in *EVALUATE* Algorithm

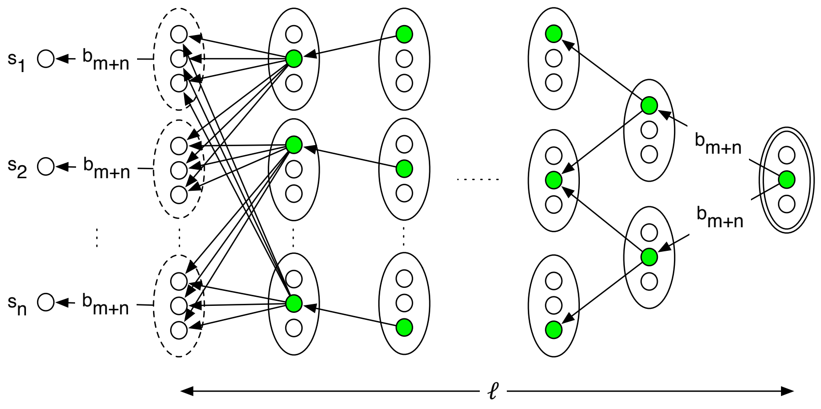

Figure 4.5: Sending back the result to the senders through a circuit of leaders in *EVALUATE* Algorithm

Algorithm 12 *EVALUATE* ▷ computes through a circuit of leaders producing the result at the output quorum, and sends back the result to all parties.

```
1: for $i = 1, \ldots, n$ do                                   ▷ provides the inputs to the circuit.
2:     Party $s_i$ calls BROADCAST $(a_i, Q_i, Q_i)$.    ▷ $s_i$ broadcasts its input $a_i$ to all parties in $Q_i$.
3:     All parties in $Q_i$ send $a_i$ to the leaders of the right neighboring quorums of $Q_i$.
4: end for
5: for $i = n+1, \ldots, m+n-1$ do                           ▷ performs the computation.
6:     Let $Q_{i'}$ and $Q_{i''}$ be the right neighboring quorums of $Q_i$ in the circuit.
7:     if leader $q_i \in Q_i$ receives all its inputs then
8:         $q_i$ performs an operation on its inputs producing an output, $b_i$.
9:         $q_i$ sends $b_i$ to leader $q_{i'} \in Q_{i'}$ and to leader $q_{i''} \in Q_{i''}$.
10:    end if
11: end for
12: if leader $q_{m+n} \in Q_{m+n}$ receives all its inputs then
13:     $q_{m+n}$ performs an operation on its inputs producing an output, $b_{m+n}$.
14:     $q_{m+n}$ broadcasts $b_{m+n}$ to all parties in $Q_{m+n}$.
15: end if
16: for $i = m+n, \ldots, n+1$ do                  ▷ sends back the result to the leftmost leaders.
17:     Let $Q_{i'}$ and $Q_{i''}$ be left neighboring quorums of $Q_i$ in circuit, for $n+1 \le i', i'' \le m+n$. *
18:     Leader $q_i \in Q_i$ sends $b_{m+n}$ to leader $q_{i'} \in Q_{i'}$ and to leader $q_{i''} \in Q_{i''}$.
19: end for
20: for $i = 1, \ldots, n$ do    ▷ sends result to all parties after broadcasting it to the input quorums.
21:     The leaders of $Q_i$'s right neighboring quorums call BROADCAST$(b_{m+n}, Q_i, Q_i)$.
22:     All parties in $Q_i$ send $b_{m+n}$ to sender $s_i$.
23: end for
```

* Recall that there are no leaders in the input quorums.

Moreover, we use *BROADCAST* in *EVALUATE* and *CHECK* in order to handle any accusation issued in *RECOVER* against the parties that provide the inputs to the input quorums, or those that receive the result in the output quorum.

Algorithm 13 *TRIGGER-CHECK* ▷ The parties of the output quorum Q_{m+n} trigger *CHECK* with probability $1/(\log^* m)^2$.

```
1: Each party in $Q_{m+n}$ chooses an input: a real number uniformly distributed between 0 and 1.
2: The parties of $Q_{m+n}$ perform MPC to find the output, prob, which is the sum of all their inputs
   modulo 1.                                   ▷ prob is the fractional part of the sum of their inputs.
3: if prob $\le 1/(\log^* m)^2$ then
4:     CHECK
5: end if
```

Our model does not directly consider concurrency. In a real system, concurrent executions of *COMPUTE* that overlap at a single quorum may allow the adversary to achieve multiple corruptions at the cost of a single marked bad party. However, this does not effect correctness, and, in practice, this issue can be avoided by serializing concurrent executions of *COMPUTE*. For simplicity of presentation, we leave the concurrency aspect out of this research.

4.6.3 *CHECK*

In this section, we describe *CHECK* algorithm, which is stated formally as Algorithm 14. In this algorithm, we make use of subquorums, where a subquorum is a subset of unmarked parties in a quorum. Let U_k be the set of all unmarked parties in quorum Q_k, for $1 \le k \le m+n$.

CHECK runs for $O(\log^* m)$ rounds. For each round i, the parties of the output quorum Q_{m+n} elect an unmarked party $\mathbf{r}$ from Q_{m+n} to be in charge of the recomputation in round i, where this election process is stated formally in *ELECT* (Algorithm 15). Then, the elected party $\mathbf{r}$ calls *REQUEST* (Algorithm 16) to send a request through a DAG of subquorums, S_j^A's, to the n senders in order to recompute (See Figure 4.6).

The recomputation process is stated formally as *RECOMPUTE* (Algorithm 17) and is shown in Figure 4.7. In this process, each sender that receives this request provides its input to redo the computation through a DAG of subquorums, S_j^B's, producing the result at the output quorum. When $\mathbf{r}$ receives this result, it calls *RESEND* (Algorithm 18) in order to send the result back to the senders through a DAG of subquorums S_j^C's, for $n+1 \leq j \leq m+n$ (See Figure 4.8).

Algorithm 14 *CHECK* ▷ Party $\mathbf{r}$ calls *CHECK* to check for corruptions.

Declaration: Let U_k be the set of all unmarked parties in quorum Q_k, for $1 \leq k \leq m+n$. Also let m' be the maximum number of parties in any quorum. Further, let subquorums, S_j^A, S_j^B and S_j^C, be initially empty, for all $n+1 \leq j \leq m+n$.

```
1: for $i \leftarrow 1, \dots, 14(\log^* m + 2(\log c + 1))^*$ do
2:     ELECT $(Q_{m+n})$                    ▷ elects an unmarked party $\mathbf{r} \in Q_{m+n}$.
3:     Party $\mathbf{r}$ constructs $A^i$, $B^i$ and $C^i$ to be three, $m$ by $m'$, arrays of random integers.**
4:     REQUEST $(i, A^i, B^i)$              ▷ $\mathbf{r}$ requests all senders to recompute.
5:     RECOMPUTE                            ▷ recomputes, producing the result, $b^i_{m+n}$, at $\mathbf{r}$.
6:     RESEND $(i, C^i, b^i_{m+n})$         ▷ $\mathbf{r}$ sends back $b^i_{m+n}$ to all parties.
7: end for
```

* $c = \frac{2(1+2p)}{\log e(1-2p)^2}$; note that for any quorum Q_k, $p \leq 1/2 - \epsilon$, is the probability of selecting a bad party u.a.r. from U_k, for any constant $\epsilon > 0$.

** $A^i[k,k']$, $B^i[k,k']$ and $C^i[k,k']$ are uniformly random integers between 1 and k', for $1 \leq k \leq m$ and $1 \leq k' \leq m'$.

Note that: if a party has previously received k_p, then it verifies each subsequent message with it; also if a party receives inconsistent messages or fails to receive and verify an expected message, then it initiates a call to *RECOVER*.

Algorithm 15 *ELECT* (Q) ▷ Parties in Q elect an unmarked party in Q using MPC.

```
1: Let each party in the set of unmarked parties, $U \subset Q$, is assigned a unique integer $\in [0, |U|)$.
2: Each party in $Q$ chooses an input: an integer uniformly distributed between 0 and $|U| - 1$.
3: The parties of $Q$ perform MPC to find the output: the sum of all their inputs modulo $|U|$.
4: The party in $U$ associated with this output number is the elected party.
```

Algorithm 16 *REQUEST* (i, A^i, B^i) $\triangleright$ **r** requests n senders through a DAG of subquorums, S_j^A's, for $n+1 \leq j \leq m+n$, to redo the computation.

```
1: Party $\mathbf{r}$ calls $SIGN([i, A^i, B^i, \mathbf{r}], Q_{m+n})$.    $\triangleright$ signs $[i, A^i, B^i, \mathbf{r}]$ by $Q_{m+n}$'s private key
2: Party $\mathbf{r}$ sets $REQ^i = ([i, A^i, B^i, \mathbf{r}]_{k_s}, k_p)$.    $\triangleright$ $(k_p, k_s)$ : public/private key pair of $Q_{m+n}$
3: Party $\mathbf{r}$ sends $REQ^i$ to all parties of quorum $Q_{m+n}$.
4: All parties in $Q_{m+n}$ calculate party, $q^i_{m+n} \in U_{m+n}$, of index $A^i_{m+n}$ to be added to $S^A_{m+n}$.*
5: for $j \leftarrow m+n, \ldots, n+1$ do    $\triangleright$ sends $REQ^i$ through a DAG of subquorums.
6:     Let $Q_{j'}$ and $Q_{j''}$ be the left neighboring quorums of $Q_j$ in the circuit, for $n+1 \leq j', j'' \leq m+n$. **
7:     All $i$ parties in $S^A_j$ calculate parties, $q^i_{j'}$ and $q^i_{j''}$, of indices $A^i_{j'}$ and $A^i_{j''}$, to be added to $S^A_{j'}$ and $S^A_{j''}$ respectively.
8:     Party $q^i_j$ calculate all parties in $S^A_{j'}$ and $S^A_{j''}$ using $A^1_{j'}, \ldots, A^i_{j'}$ and $A^1_{j''}, \ldots, A^i_{j''}$.
9:     for $k \leftarrow 1, \ldots, i$ do    $\triangleright$ $k$ refers to the rounds prior to round $i$.
10:        Party $q^k_j$ sends $REQ^k$ to parties $q^i_{j'}$ and $q^i_{j''}$.
11:        Party $q^i_j$ sends $REQ^i$ to parties $q^k_{j'}$ and $q^k_{j''}$.
12:    end for
13: end for
14: for $k \leftarrow n, \ldots, 1$ do    $\triangleright$ The input quorums forward $REQ^i$ to all senders.
15:    Let $Q_{k'}$ and $Q_{k''}$ be the right neighboring quorums of $Q_k$ in the circuit.
16:    All $i$ parties in $S_{k'}$ and all parties in $S_{k''}$ call $BROADCAST(REQ^i, Q_k, Q_k)$.
17:    All parties in $Q_k$ send $REQ^i$ to sender $s_k$.
18: end for
```

* $A^i_j = A^i[j-n, |U_j|]$ is the index of the party, q^i_j, which is selected u.a.r. from the parties in U_j in round i of *REQUEST*; note that all parties in U_j are sorted by their IDs, for $n+1 \leq j \leq m+n$.

** Recall that there are no subquorums for the input quorums.

Algorithm 17 *RECOMPUTE* ▷ n senders provide inputs to a DAG of subquorums, S_j^B's, for $n+1 \le j \le m+n$, to recompute, producing a result, b_{m+n}^i, at $\mathbf{r}$.

```
 1: for each sender $s_j$ that receives $REQ^i$, for $1 \le j \le n$ and $n+1 \le j', j'' \le m+n$ do
 2:     $s_j$ sets $REC^i$ to be a message consisting of its input $a_j$ and $REQ^i$.
 3:     $s_j$ broadcasts $REC^i$ to all parties in $Q_j$.
 4:     Let $Q_{j'}$ and $Q_{j''}$ be the right neighboring quorums of $Q_j$ in the circuit.
 5:     All parties in $Q_j$ calculate parties, $q_{j'}^i$ and $q_{j''}^i$, of indices $B_{j'}^i$ and $B_{j''}^i$, to be added to $S_{j'}^B$
    and $S_{j''}^B$ respectively.*
 6:     All parties in $Q_j$ send $REC^i$ to all parties in $S_{j'}^B$ and to all parties in $S_{j''}^B$.
 7:     All parties in $Q_j$ send $REC^1, \ldots, REC^{i-1}$ to $q_{j'}^i$ and $q_{j''}^i$.
 8: end for
 9: for $j \leftarrow n+1, \ldots, m+n-1$ do                                          ▷ recomputes
10:     Let $Q_{j'}$ and $Q_{j''}$ be the right neighboring quorums of $Q_j$ in the circuit.
11:     All $i$ parties in $S_j^B$ calculate parties, $q_{j'}^i$ and $q_{j''}^i$, of indices $B_{j'}^i$ and $B_{j''}^i$, to be added to $S_{j'}^B$
    and $S_{j''}^B$ respectively.
12:     Party $q_j^i$ calculate all parties in $S_{j'}^B$ and $S_{j''}^B$ using $B_{j'}^1, \ldots, B_{j'}^i$ and $B_{j''}^1, \ldots, B_{j''}^i$.
13:     for all $1 \le k \le i$, $q_j^k$ performs its operation on its inputs producing an output, $b_j^k$.
14:     for $k \leftarrow 1, \ldots, i$ do
15:         $q_j^k$ sends $b_j^k$ and $REC^k$ to parties $q_{j'}^i$ and $q_{j''}^i$.
16:         $q_j^i$ sends $b_j^i$ and $REC^i$ to parties $q_{j'}^k$ and $q_{j''}^k$.
17:     end for
18: end for
19: All $i$ parties in $S_{m+n}$ broadcast $b_{m+n}^i$ and $REC^i$ to all parties in $Q_{m+n}$.
20: All parties in $Q_{m+n}$ send $b_{m+n}^i$ and $REC^i$ to party $\mathbf{r}$.          ▷ $\mathbf{r}$ receives the result.
```

* $B_j^i = B^i[j-n, |U_j|]$ is the index of the party, q_j^i, which is selected u.a.r. from the parties in U_j in round i of *RECOMPUTE*; note that all parties in U_j are sorted by their IDs, for $n+1 \le j \le m+n$.

Note that in *ELECT* (Q), the parties of quorum Q perform MPC [6, 17, 68] to elect an unmarked party, $\mathbf{r}$, uniformly at random from Q. We know that at least half of the unmarked parties in Q are good. Thus, the elected party is good with probability at least 1/2. MPC requires a message cost and a number of computational operations that are polylogarithmic functions in n, and it runs in $O(1)$ time.

Algorithm 18 *RESEND* (i, C^i, b^i_{m+n}) $\triangleright$ Party $\mathbf{r}$ sends back the result, b^i_{m+n}, through a DAG of subquorums, S^C_j's, to n senders, for $n+1 \le j \le m+n$.

```
 1: Party $\mathbf{r}$ calls $SIGN([i, C^i, b^i_{m+n}, \mathbf{r}], Q_{m+n})$.          $\triangleright$ signs it by $Q_{m+n}$'s private key.
 2: Party $\mathbf{r}$ sets $RES^i = ([i, C^i, b^i_{m+n}, \mathbf{r}]_{k_s}, k_p)$.   $\triangleright$ $(k_p, k_s)$ : public/private key pair of $Q_{m+n}$.
 3: Party $\mathbf{r}$ sends $RES^i$ to all parties of quorum $Q_{m+n}$.
 4: All parties in $Q_{m+n}$ calculate party, $q^i_{m+n} \in U_{m+n}$, of index $C^i_{m+n}$ to be added to $S^C_{m+n}$.*
 5: for $j \leftarrow m+n, \ldots, n+1$ do          $\triangleright$ sends back the result through a DAG of subquorums.
 6:     Let $Q_{j'}$ and $Q_{j''}$ be the left neighboring quorums of $Q_j$ in the circuit, for $n+1 \le j', j'' \le m+n$. **
 7:     All $i$ parties in $S^C_j$ calculate parties, $q^i_{j'}$ and $q^i_{j''}$, of indices $C^i_{j'}$ and $C^i_{j''}$, to be added to $S^C_{j'}$ and $S^C_{j''}$ respectively.
 8:     Party $q^i_j$ calculate all parties in $S^C_{j'}$ and $S^C_{j''}$ using $C^1_{j'}, \ldots, C^i_{j'}$ and $C^1_{j''}, \ldots, C^i_{j''}$.
 9:     for $k \leftarrow 1, \ldots, i$ do          $\triangleright$ $k$ refers to the rounds prior to round $i$.
10:         Party $q^k_j$ sends $RES^k$ to parties $q^i_{j'}$ and $q^i_{j''}$.
11:         Party $q^i_j$ sends $RES^i$ to parties $q^k_{j'}$ and $q^k_{j''}$.
12:     end for
13: end for
14: for $k \leftarrow n, \ldots, 1$ do          $\triangleright$ The input quorums forward $RES^i$ to all senders.
15:     Let $Q_{k'}$ and $Q_{k''}$ be the right neighboring quorums of $Q_k$ in the circuit.
16:     All $i$ parties in $S_{k'}$ and all parties in $S_{k''}$ call $BROADCAST(RES^i, Q_k, Q_k)$.
17:     All parties in $Q_k$ send $RES^i$ to sender $s_k$.
18: end for
```

* $C^i_j = C^i[j-n, |U_j|]$ is the index of the party, q^i_j, which is selected u.a.r. from the parties in U_j in round i of *RESEND*; note that all parties in U_j are sorted by their IDs, for $n+1 \le j \le m+n$.
** Recall that there are no subquorums for the input quorums.

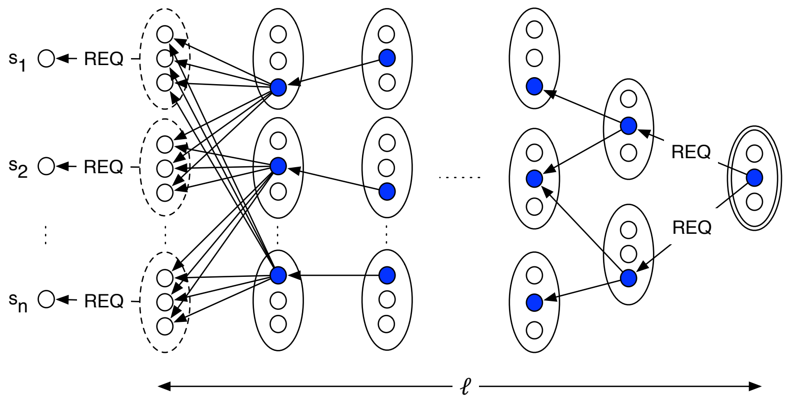

Figure 4.6: *REQUEST* Algorithm

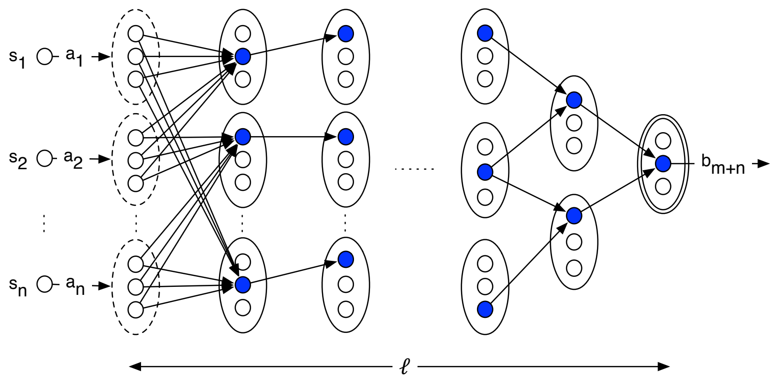

Figure 4.7: *RECOMPUTE* Algorithm

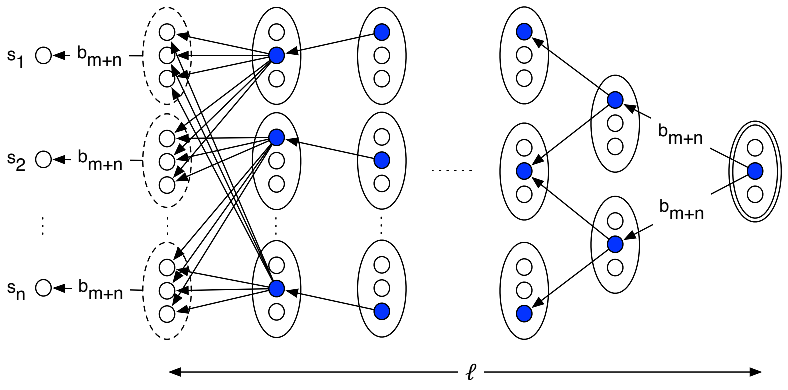

Figure 4.8: *RESEND* Algorithm

Note further that during *CHECK*, if any party receives inconsistent messages or fails to receive and verify any expected message in any round, it initiates a call to *RECOVER*.

CHECK detects message corruptions with probability at least 1/2. It requires $O((m + n\log n)(\log^* n)^2)$ message cost and $O(\ell \log^* n)$ latency. But since *CHECK* is triggered with probability $1/(\log^* m)^2$, it has expected message cost $O(m + n\log n))$ with expected latency $O(\ell/\log^* n)$.

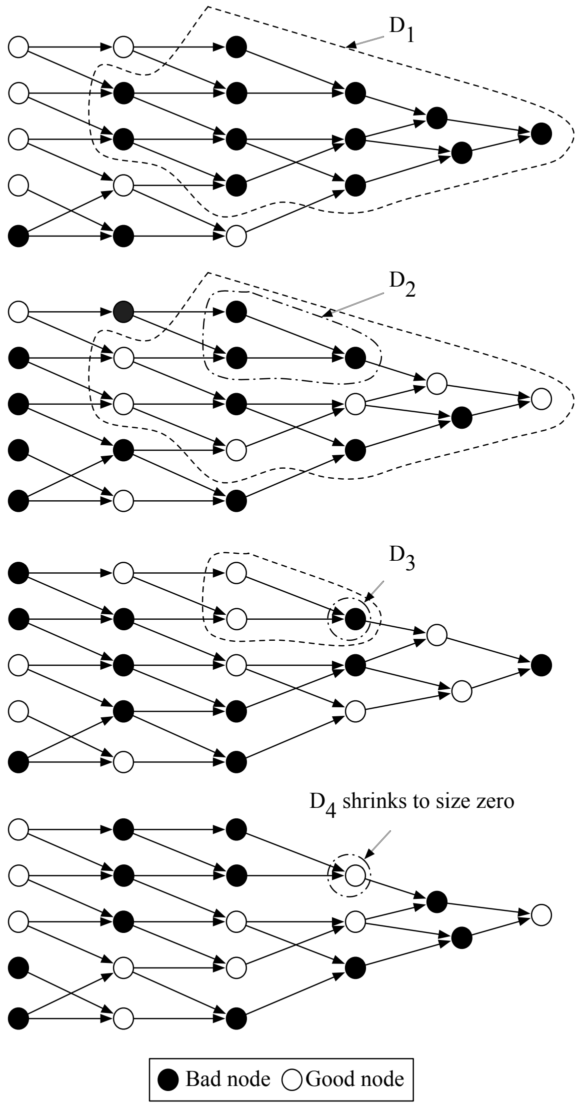

Figure 4.9: Example run of *CHECK*

An example run of *CHECK* is illustrated in Figures 4.9, 4.10 and 4.11. These figures show that in each round, a circuit of parties is formed, where one party is selected u.a.r. from each quorum in the quorum graph. Each of these parties performs the appropriate gate operation on its inputs providing an output which is an input for the next gate in the circuit.

For a given circuit of parties and a given round, there is a white or black node depending on whether the party selected in that particular round and that particular gate (quorum) is good (white) or bad (black). Informally, we define a *deception DAG*, D_i, in a round, i, as a rooted DAG of bad nodes that are selected in this round to be added to the subquorums in the quorum graph.

In Figure 4.9, we show the deception DAGs that are chosen by the adversary to corrupt the computation over rounds. In particular, the adversary's strategy is 1) to corrupt the output of the maximum deception DAG in the first round; and 2) to keep corrupting this output by nesting levels of deception DAGs in all subsequent rounds. These nesting levels of deception DAGs are outlined in this figure.

There are two key points by which *CHECK* detects corruptions: 1) any deception DAG in any round never extends in any subsequent round; and 2) any deception DAG is expected to shrink logarithmically in size from round to round. This will imply that any deception DAG shrinks to size zero after $O(\log^* n)$ rounds, at which the corruption is detected.

Deception DAGs never extend? We know that each good party receives an input message in any round has to receive the same input message in all subsequent rounds; otherwise, it will call *RECOVER*. Moreover, for each round, we know that all parties in each subquorum send their output message to the new party that is added in this round to the next subquorum in the circuit. Thus, those good parties that provide their output message to the deception DAG of this round will provide the same output message to all subsequent deception DAGs. Recall that in each round, every party that is added to each subquorum, S, sends its output message to all parties in the next subquorum, S', in the quorum graph.

Thus, all good parties in S' that receive an input message through a deception DAG in any round expect to receive the same input message in all subsequent rounds. Also, each good party that did not receive the correct input message in any round must not receive this message in all subsequent rounds; otherwise, it will initiate a call to *RECOVER*.

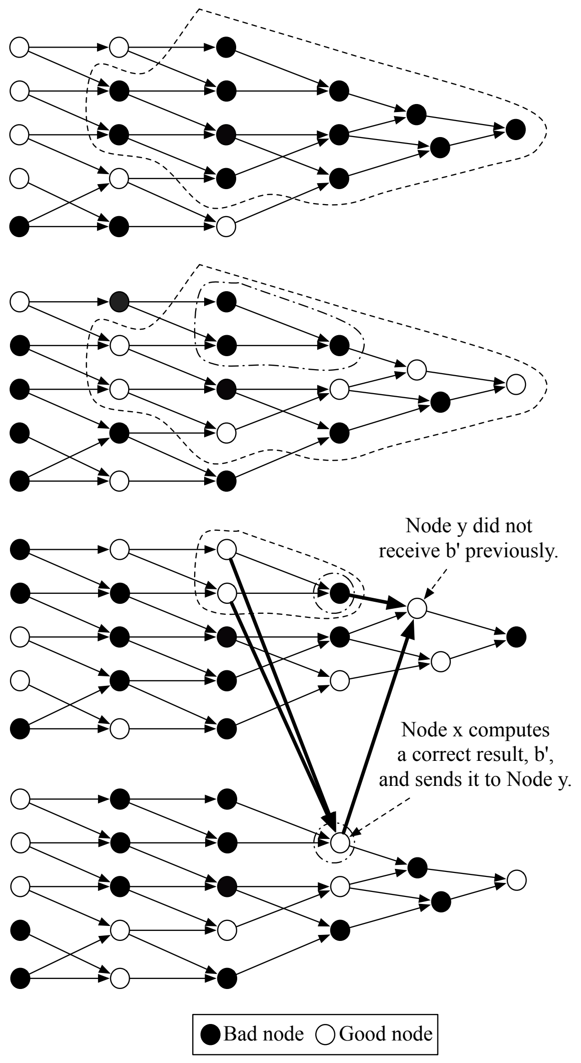
Node y did not receive b' previously.
Node x computes a correct result, b', and sends it to Node y.
Bad node
Good node

Figure 4.10: A corruption is detected after the deception DAG shrinks to size zero.

Deception DAGs shrink logarithmically? A key lemma (Lemma 4.7.4) shows that given a rooted DAG of size m, in which each bad node is selected u.a.r. with probability at most 1/2, then the probability of having a rooted subgraph of $\Omega(\log m)$ bad nodes is at most 1/2. Intuitively, we could expect that $O(\log^* m)$ rounds will suffice to shrink any deception DAG to size zero in a quorum graph of m gates.

When any deception DAG shrinks to size zero, is corruption detected? Figure 4.10 shows that when the deception DAG shrinks over rounds to size zero, node x in the last round receives correct input messages. Then node x computes a correct output and sends it to node y; however, node y has not previously received it as an input in this call to *CHECK*. As a result, node y calls *RECOVER* declaring that it has received inconsistent input messages.

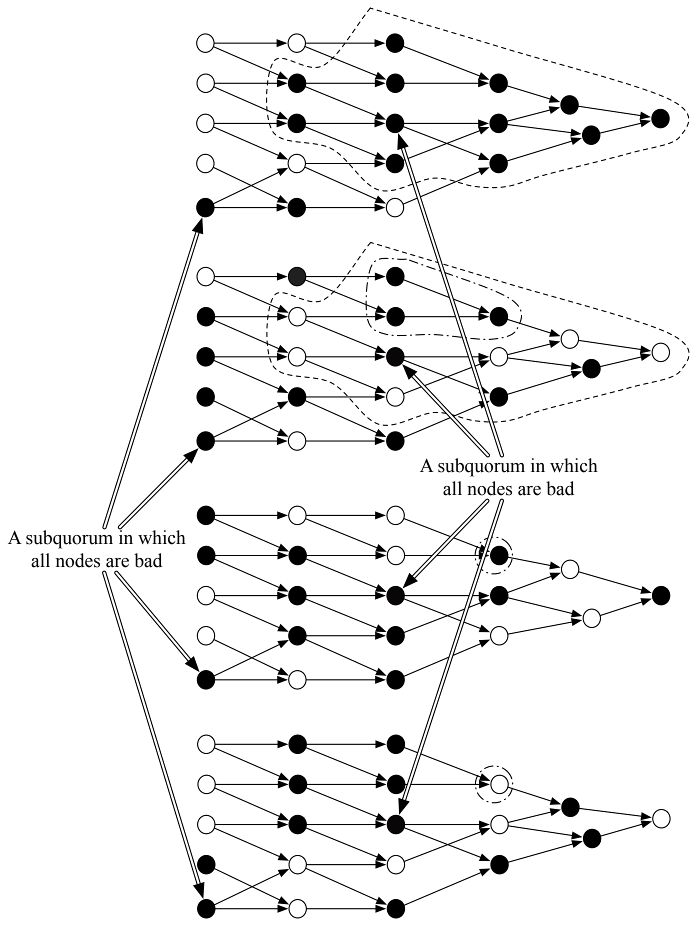
A subquorum in which all nodes are bad
A subquorum in which all nodes are bad

Figure 4.11: A corruption can be detected even if all nodes in multiple subquorums are bad.

If all nodes in multiple subquorums are bad, does *CHECK* successfully detect message corruptions? *CHECK* can detect corruptions even if all parties in some subquorums are bad. Recall that *CHECK* runs in $O(\log^* m)$ rounds. In each round, new parties are selected u.a.r. to be added to the subquorums. This limits the adversary to know, before all rounds finish, if all parties of any particular subquorum are bad. Thus, the adversary would rather corrupt the output of the maximum deception DAG in the first round and keeps corrupting this output over all subsequent deception DAGs. Note that if the adversary corrupts more than one output in the same round, it will increase the chance of detecting corruptions. Figure 4.11 shows that even though all parties in multiple subquorums are bad, most of these parties behave as good parties since they are out of the deception DAGs selected by the adversary.

4.6.4 *RECOVER*

When a computation is corrupted and *CHECK* detects this corruption, *RECOVER* is called. The *RECOVER* algorithm is described formally as Algorithm 19. When *RECOVER* starts, all parties in each quorum in the circuit are notified.

The main purpose of *RECOVER* is to 1) determine the location in which the corruption occurred; and 2) mark the parties that are in conflict.

Algorithm 19 *RECOVER* ▷ Party $q' \in Q'$ calls *RECOVER* after it detects a corruption.

```
1: $q'$ broadcasts to all parties in $Q'$ the fact that it calls RECOVER along with the messages it
   has received in this call to COMPUTE.
2: The parties in $Q'$ verify that $q'$ received inconsistent messages before proceeding.
3: $Q'$ notifies all quorums in the circuit via all-to-all communication that RECOVER is called.
4: INVESTIGATE          ▷ investigates all participants to determine corruption locations.
5: MARK-IN-CONFLICTS    ▷ marks the parties that are in conflict.
```

Algorithm 20 *INVESTIGATE* ▷ investigates the parties that have participated.

```
1: for each party, $q$, involved in the last call to $EVALUATE$ or $CHECK$ do
2:     $q$ compiles all messages they have received (and from whom) and they have sent (and to whom) in the last call to $EVALUATE$ or $CHECK$.
3:     $q$ broadcasts these messages to all parties in its quorum and neighboring quorums.
4: end for
```

Algorithm 21 *MARK-IN-CONFLICTS* ▷ marks the parties that are in conflict.

```
1: for each pair of parties, $(q_x, q_y)$, that is in conflict*, in quorums $(Q_x, Q_y)$ do
2:     party $q_y$ broadcasts a conflict message, $\{q_x, q_y\}$, to all parties in $Q_y$.
3:     each party in $Q_y$ forwards $\{q_x, q_y\}$ to all parties in $Q_x$.
4:     all parties in $Q_x$ (or $Q_y$) send $\{q_x, q_y\}$ to the other quorums that has $q_x$ (or $q_y$).
5:     each quorum has $q_x$ or $q_y$ sends $\{q_x, q_y\}$ to its neighboring quorums.
6: end for
7: for each party $q$ that receives conflict message $\{q_x, q_y\}$ do
8:     $q$ marks $q_x$ and $q_y$ in its marking table.
9: end for
10: if $(1/2 - \gamma)$-fraction of parties in any quorum have been marked, for $\gamma = 0.01$ then
11:     each of these parties is set unmarked in all its quorums.
12:     each of these parties is set unmarked in all its neighboring quorums.
13: end if
14: for each pair of leaders, $(q_x, q_y)$, that is in conflict, in quorums $(Q_x, Q_y)$ do
15:     ELECT($Q_x$) and ELECT($Q_y$) to elect a pair of unmarked leaders, $(q'_x, q'_y)$.
16:     $Q_x$ and $Q_y$ notify their neighboring quorums with $(q'_x, q'_y)$.
17: end for
```

* A pair of parties, (q_x, q_y), is *in conflict* if: 1) q_x was scheduled to send an output to q_y at some point in the last call to *EVALUATE* or *CHECK*; and 2) q_y does not receive an expected message from q_x in *INVESTIGATE*, or q_y receives a message in *INVESTIGATE* that is different than the message that it has received from q_x in the last call to *EVALUATE* or *CHECK*.

To determine the location in which the corruption occurred, *RECOVER* calls *INVESTIGATE* (Algorithm 20) to investigate the corruption situation by letting each party involved in *EVALUATE* or *CHECK* broadcast all messages they have received or sent. Then, *RECOVER* calls *MARK-IN-CONFLICTS* (Algorithm 21) in order to mark the parties that are *in conflict*, where a pair of parties is in conflict if at least one of these parties broadcasted messages that conflict with the messages broadcasted by the other party in this pair.

Note that each pair of parties that are in conflict has at least one bad party. Recall that if $(1/2 - \gamma)$-fraction of parties in any quorum are marked, for any constant $\gamma > 0$, e.g., $\gamma = 0.01$, they are set unmarked. Also, for each pair of leaders that get marked, their quorums elect another pair of unmarked leaders.

4.7 Analysis

In this section, we prove the lemmas required for Theorem 4.3.1. Throughout this section, all logarithms are base 2.

Recall that in each round of *CHECK*, a new unmarked party is selected u.a.r. from each quorum in the circuit forming a new DAG of unmarked parties.

Definition 4.7.1. *A* Deception DAG*, D_i, is the maximal subgraph of the new DAG of unmarked parties that are selected u.a.r. in round i, with the following properties: 1) it has only bad parties; 2) it receives all its inputs, and each input is provided correct by at least one good party; 3) it is rooted at one party, which does not provide a correct output to at least one good party; and 4) all other outputs of this DAG are provided correct.*

If the adversary corrupts the output of the root party in a deception DAG in any round, then it has to keep corrupting this output by a deception DAG in each subsequent round; otherwise, the good parties that expect to receive this output in each round will call *RECOVER* due to receiving inconsistent output messages.

We say that a deception DAG, D_i, in round i extends in round $i+1$ if there exists a deception DAG, D_{i+1}, in round $i+1$ such that 1) there is at least one subquorum that has a party in D_i and a party in D_{i+1}; and 2) there is at least one subquorum that has a party in D_{i+1} but has no party in D_i.

Also, we say that a deception DAG, D_i, in round i shrinks in round $i+1$ if there exists a deception DAG, D_{i+1}, in round $i+1$ such that 1) each subquorum that has a party in D_{i+1} has a party in D_i; and 2) there is at least one subquorum that has a party in D_i but has no party in D_{i+1}.

Further, we say that a deception DAG, D_i, shrinks logarithmically from round i to round $i+1$ if $|D_{i+1}| = O(\log |D_i|)$.

4.7.1 *CHECK*

In the following lemmas, we first show that any deception DAG in any round never extends in any subsequent round. Then we show that with probability at least 1/2, any deception DAG shrinks logarithmically from round to round. This will imply that the expected number of rounds to shrink any deception DAG to size zero is $O(\log^* m)$.

Note that in any round i, if a deception DAG, D_i, shrinks to a deception DAG, D_{i+1}, of size zero in round $i+1$, then the good party that did not receive the correct output from D_i in round i will receive the correct output in round $i+1$. As a result, this good party will call *RECOVER* declaring that it has received inconsistent output messages.

Lemma 4.7.2. *Any deception DAG in any round never extends.*

Proof. We know by definition that any deception DAG is confined by 1) the good parties that provide the inputs to this deception DAG; and 2) the good parties that receive the outputs from it.

In each round i, all i parties in each subquorum send their outputs to the new party that is added to the next subquorum in this round. Thus, the good parties that provide the correct inputs to a deception DAG in round i, will provide the correct inputs to all nesting level of deception DAGs in all subsequent rounds.

Moreover, in each round i, each new party that is added to a subquorum in this round sends its output to all i parties in the next subquorum. Thus, the good parties that receive an output from a deception DAG in round i, must receive the same output from all nesting level of deception DAGs in all subsequent rounds. Similarly, the good parties that did not receive the correct output from a deception DAG in round i must not receive this output from any nesting level of deception DAG in any subsequent round; otherwise, they will call *RECOVER*.

Further, if a good party has previously received k_p (the public key of Q_{m+n}), then it verifies each subsequent message with it; also if a party receives inconsistent messages or fails to receive and verify an expected message, then it initiates a call to *RECOVER*.

Therefore, all good parties that confine a deception DAG in any round will restrict all subsequent deception DAGs. $\square$

Now we show that any deception DAG shrinks logarithmically from round to round with probability at least 1/2.

Definition 4.7.3. Rooted Directed Acyclic Graph (R-DAG) *is a DAG in which, for a vertex u called the root and any other node v, there is at least one directed path from v to u.*

Lemma 4.7.4. *Given any R-DAG, of size n, in which each node has indegree of at most d and survives independently with probability at most p such that $0 < p \leq \frac{1}{d} - \epsilon$, for any constant $\epsilon > 0$, then the probability of having a subgraph, rooted at some node, with surviving nodes, of size $\Omega(\frac{\log n}{(1-pd)^2})$ is at most $1/2$.*

Proof. This proof has three propositions. In this proof, we recall the exploration process that was introduced in [44, 51].

Given an R-DAG, $D(V, E)$ comprising a set V of nodes and a set E of edges, with size $n = |V|$ and maximum indegree d. After each node survives independently with probability at most p such that $0 < p \leq \frac{1}{d} - \epsilon$, for any constant $\epsilon > 0$, we explore D to find a subgraph of surviving nodes, of size more than k, rooted at an arbitrary node v (assuming that node v survives).

Let $D'(v)$ be the maximal subgraph of surviving nodes, rooted at node v. Let each node in D have a status, which is either *inactive*, *active* or *neutral*. During the exploration process, the status of any node can be changed. A node is *inactive* if this node and its children are explored. A node is *active* if this node is explored and its children are not explored yet. A node is *neutral* if it is neither active nor inactive, i.e., this node and its children are not explored yet.

We let the exploration process run at most $k \geq 0$ steps. Initially, we select an arbitrary surviving node, v. We set the node v active and all other nodes neutral. At each step i, we choose an active node, w_i, in an arbitrary way, and we explore all its children as follows. For all $(w_i, w_i') \in E$ such that w_i' is neutral and survives, we set w_i' active, otherwise w_i' remains as it is. After we explore all children of w_i, we set w_i inactive. Note that at any step, if there is no active node, the exploration process terminates.

Now let d_i be the number of children of node w_i for $1 \leq i \leq k$. For $1 \leq i \leq k$, let X_i be a non-negative random variable for the number of surviving neutral children of w_i, and let Y_i be a non-negative random variable for the number of surviving non-neutral children of w_i. Note that $Y_1 = 0$. Hence, $X_i \sim Bin(d_i - Y_i, p)$ if the exploration process does not terminate before i steps; otherwise, $X_i = 0$. This implies that $X_i \sim Bin(d_i - Y_i, p_i)$ for some $0 \leq p_i \leq p$. Let A_i be a non-negative random variable for the total number of active nodes after i steps, for $1 \leq i \leq k$.

Proposition 4.7.5. $A_i = \begin{cases} \sum_{i=1}^{k} X_i - (k-1) & \text{if } A_{i-1} > 0, \\ 0 & \text{otherwise.} \end{cases}$

Proof. Since the process starts initially with one active node v, we have $A_0 = 1$. Now we have two cases of A_{i-1} to compute A_i, $1 \leq i \leq k$:

Case 1 (process terminates before i steps): If $A_{i-1} = 0$, then $A_i = 0$.

Case 2 (otherwise): If $A_{i-1} > 0$, then $A_i = A_{i-1} + X_i - 1$, where after exploring w_i, the total number of active nodes is the number of new active nodes (X_i) due to the exploration of w_i plus the total number of active nodes of previous steps (A_{i-1}), excluding w_i that becomes inactive at the end of step i. □

Now let $|D'(v)|$ be the number of nodes in $D'(v)$.

Proposition 4.7.6. $Pr(|D'(v)| > k) \leq Pr(\sum_{i=1}^{k} X_i \geq k)$.

Proof. To prove this proposition, we first prove that $Pr(|D'(v)| > k) \leq Pr(A_k > 0)$, or equivalently, we prove that $|D'(v)| > k \implies A_k > 0$. If $|D'(v)| > k$, then the exploration process does not terminate before k steps. This implies that after k steps, there are k inactive nodes and at least one active node remains. This follows that $A_k > 0$. Thus, we have

$$Pr(|D'(v)| > k) \leq Pr(A_k > 0). \tag{4.1}$$

Second, we prove that $Pr(A_k > 0) \leq Pr(\sum_{i=1}^{k} X_i - (k-1) > 0)$, or equivalently, we prove that $A_k > 0 \implies \sum_{i=1}^{k} X_i - (k-1) > 0$. If $A_k > 0$, then $A_j > 0$ for all $1 \leq j \leq k$. By Proposition 4.7.5, we obtain that $\sum_{i=1}^{j} X_i - (j-1) > 0$ for all $1 \leq j \leq k$. This follows that

$$Pr(A_k > 0) \leq Pr(\sum_{i=1}^{k} X_i - (k-1) > 0). \tag{4.2}$$

By Inequalities (4.1) and (4.2), we obtain

$$Pr(|D'(v)| > k) \leq Pr(\sum_{i=1}^{k} X_i - (k-1) > 0),$$

or equivalently,

$$Pr(|D'(v)| > k) \leq Pr(\sum_{i=1}^{k} X_i > k-1).$$

Since k is a positive integer, we have

$$Pr(|D'(v)| > k) \leq Pr(\sum_{i=1}^{k} X_i \geq k).$$

□

Proposition 4.7.7. $Pr(\sum_{i=1}^{k} X_i \geq k) \leq e^{-\frac{(1-pd)^2 k}{1+pd}}$.

Proof. Likewise the proof of giant component in [41], for $1 \leq i \leq k$, let $X_i^+ \sim Bin(d,p)$, and let $X_1^+, ..., X_k^+$ be independent random variables. We know that $Y_i \geq 0$ and $d_i \leq d$ for $1 \leq i \leq k$.

Recall that for all i, for each event $\xi = (X_1 = x_1, ..., X_{i-1} = x_{i-1})$ where $x_1, ..., x_{i-1} \in \{0, ..., d\}$, $X_i \sim Bin(d_i - Y_i, p_i)$, for some $0 \leq p_i \leq p$.

With inequalities in terms of stochastic dominance [15, 36, 47, 78], for all i, $X_i \leq_{st} X_i^+$.

Thus, we have

$$Pr(\sum_{i=1}^{k} X_i \geq k) \leq Pr(\sum_{i=1}^{k} X_i^+ \geq k).$$

Now let $S_k = \sum_{i=1}^{k} X_i^+$. By Chernoff bounds [53, 56, 63], for $\delta > 0$, we obtain

$$Pr(S_k \geq (1+\delta)\mathbf{E}(S_k)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mathbf{E}(S_k)} \leq e^{-\frac{\delta^2}{2+\delta}\mathbf{E}(S_k)}.$$

We know that $S_k \sim Bin(kd, p)$. Thus, $\mathbf{E}(S_k) = pdk$. Therefore, we have

$$Pr(S_k \geq (1+\delta)pdk) \leq e^{-\frac{\delta^2}{2+\delta}pdk}.$$

For $\delta = \frac{1-pd}{pd}$, we obtain

$$Pr(S_k \geq k) \leq e^{-\frac{(1-pd)^2 k}{1+pd}}.$$

□

Now by Propositions 4.7.6 and 4.7.7, we have

$$Pr(|D'(v)| > k) \leq e^{-\frac{(1-pd)^2 k}{1+pd}}.$$

We know that node v survives with probability at most p. Thus, we obtain

$$Pr(|D'(v)| > k) \leq pe^{-\frac{(1-pd)^2 k}{1+pd}}.$$

Union bound over n nodes, then the probability that there exists a subgraph of D, rooted at one node, having only surviving nodes, of size more than k is at most

$$nPr(|D'(v)| > k) \leq npe^{-\frac{(1-pd)^2 k}{1+pd}}.$$

Note that $n\ p\ e^{-\frac{(1-pd)^2 k}{1+pd}} \leq 1/2$ when $k \geq \frac{1+pd}{(1-pd)^2 \log e} \log(2pn)$. Thus, the probability of having such a subgraph of size more than $\frac{1+pd}{(1-pd)^2 \log e} \log(2pn)$, or equivalently, $\Omega\left(\frac{\log n}{(1-pd)^2}\right)$, is at most 1/2. □

Corollary 4.7.8. *For any R-DAG, of size n, the probability of having a subgraph, rooted at one node, with surviving nodes, of size at least $n/2$ is at most $1/2$.*

Now, if a deception DAG shrinks logarithmically in a successful step, then how many successful steps to shrink this deception DAG to a deception DAG of size zero or even of a constant size?

Let $f(n) = c \log n$, and let $f^{(i)}(n)$ be the function of applying function f, i times, over n. Also, we let $\log^{(i)}(n)$ be the function of applying logarithm i times over n.

Fact 4.7.9. $\forall i \geq 1 : \log^{(i)}(n) \geq \log c + 1,\ f^{(i)}(n) \leq 2c \log^{(i)}(n)$.

Proof. We prove by induction over $i \geq 1$ that for $\log^{(i)}(n) \geq \log c + 1$,

$$f^{(i)}(n) \leq 2c \log^{(i)}(n).$$

Base case: for $i = 1$, by definition,

$$f(n) = c \log n \leq 2c \log n.$$

Induction hypothesis: for $\log^{(j)}(n) \geq \log c + 1$,

$$\forall j < i, f^{(j)}(n) \leq 2c \log^{(j)}(n).$$

Induction step: by definition,

$$f^{(i)}(n) = f(f^{(i-1)}(n)).$$

By induction hypothesis, for $\log^{(i-1)}(n) \geq \log c + 1$,

$$f^{(i-1)}(n) \leq 2c \log^{(i-1)}(n).$$

Then,

$$f^{(i)}(n) \leq f(2c \log^{(i-1)}(n)) = c \log(2c \log^{(i-1)}(n)),$$

or equivalently,

$$f^{(i)}(n) \leq c(1 + \log c + \log^{(i)}(n)) \leq 2c \log^{(i)}(n),$$

for $\log^{(i)}(n) \geq \log c + 1$. □

Now let $f^*(n)$ be the smallest value i such that $f^{(i)}(n) \leq c(2c + \log c + 1)$.

Fact 4.7.10. $\forall n > c(2c + \log c + 1), f^*(n) \leq \log^* n - \log^*(\log c + 1)$.

Proof. Let $k = \log^* n - \log^* (\log c + 1) - 1$. Then, $\log^{(k)}(n) \geq \log c + 1$. By Fact 4.7.9,

$$f^{(k)}(n) \leq 2c \log^{(k)}(n).$$

With a further application of f to $f^{(k)}(n)$, we have

$$f^{(k+1)}(n) \leq c \log(2c \log^{(k)}(n)) = c(1 + \log c + \log^{(k+1)}(n)).$$

We know that $\log^{(k+1)}(n) \leq 2c$. Thus, we obtain

$$f^{(k+1)}(n) \leq c(1 + \log c + 2c).$$

Therefore, by definition,

$$f^{*}(n) \leq k + 1 = \log^{*} n - \log^{*}(\log c + 1).$$

□

Lemma 4.7.11. *Assume that any deception DAG of size n' shrinks to a deception DAG of size $c \log n'$ in a successful step, for any constant $c \geq 1$. Then, for a deception DAG of size $n > c(2c + \log c + 1)$, after $\log^{*} n - \log^{*}(\log c + 1)$ successful steps, it shrinks to a deception DAG of size at most $c(2c + \log c + 1)$.*

Proof. Fact 4.7.10 proves this lemma. □

Let p be the probability of selecting an unmarked bad party uniformly at random in any quorum. Recall that the fraction of bad parties in any quorum is at most $1/4$, and at any time the fraction of unmarked parties in any quorum is at least $1/2 + \gamma$, for any constant $\gamma > 0$. Thus, $p \leq \frac{1/2}{1+2\gamma}$.

Now we show the expected number of rounds to shrink any deception DAG to size zero.

Lemma 4.7.12. *With probability at least $1/2$, any deception DAG of size m shrinks to size zero in $14(\log^{*} m + 2(\log c + 1))$ rounds, where $c = \frac{2(1+2p)}{\log e(1-2p)^2}$ and $p \leq \frac{1/2}{1+2\gamma}$, for any constant $\gamma > 0$.*

Proof. Given a deception DAG, of size m. By Lemma 4.7.2, the deception DAG never extends over rounds. For shrinking deception DAGs over rounds, we make use of Lemma 4.7.4 to shrink logarithmically any deception DAG of size more than $c(2c + \log c + 1)$; otherwise, deception DAGs shrink geometrically using Corollary 4.7.8.

Let X_i be an indicator random variable that is equal 1 if the deception DAG in round i shrinks logarithmically in round $i+1$; and 0 otherwise.

By Lemma 4.7.11, after at most $\log^* m - 1$ rounds of X_i's equal 1, the deception DAG of size at most m shrinks to a size of at most $c(2c + \log c + 1)$.

Also let Y_j be an indicator random variable that is equal 1 if the deception DAG of size at most $c(2c + \log c + 1) \leq 4c^2$ in round j shrinks geometrically by at most half the size in round $j+1$; and 0 otherwise.

In order to shrink the deception DAG of size n to 0, we require at most $\log^* m - 1$ rounds of X_i's equal 1 and at most $2\log c + 3$ rounds of Y_j's equal 1.

Note that in each round, the receiver that is elected by the output quorum is good with probability at least 1/2. By Lemma 4.7.4, for all i, for each event $\xi = (X_1 = x_1, ..., X_{i-1} = x_{i-1})$ where $x_1, ..., x_{i-1} \in \{0, 1\}$, $X_i \sim \mathit{Bernoulli}(p_i)$ for some $p_i \geq 1/4$. Also, by Corollary 4.7.8, for all j, for each event $\xi' = (Y_1 = y_1, ..., Y_{j-1} = y_{j-1})$ where $y_1, ..., y_{j-1} \in \{0, 1\}$, $Y_j \sim \mathit{Bernoulli}(p_j)$ for some $p_j \geq 1/4$.

Let ξ_X be the event that $\sum_{i=1}^{14(\log^* m-1)} X_i \geq (\log^* m - 1)$, and let ξ_Y be the event that $\sum_{j=1}^{14(2\log c+3)} Y_j \geq 2\log c + 3$.

We know that the probability, that any deception interval of size m shrinks to size zero, is at least

$$\Pr(\xi_X \cap \xi_Y) = \Pr(\xi_Y | \xi_X) \cdot \Pr(\xi_X).$$

$$\Pr(\xi_Y|\xi_X) \cdot \Pr(\xi_X) = \Pr(\sum_{j=1}^{14(2\log c+3)} Y_j \geq 2\log c + 3) \cdot \Pr(\sum_{i=1}^{14(\log^* m-1)} X_i \geq (\log^* m - 1)).$$

For all $1 \leq k \leq 14(\log^* m + 2(\log c + 1))$, let $Z_k \sim \mathit{Bernoulli}(1/4)$, and all Z_k's are independent random variables. With inequalities in terms of stochastic dominance [15, 36, 47, 78], for all i, j, k, $X_i \geq_{st} Z_k$ and $Y_j \geq_{st} Z_k$.

$$\Pr(\xi_Y|\xi_X) \cdot \Pr(\xi_X) \geq \Pr\left(\sum_{j=1}^{14(2\log c+3)} Z_j \geq 2\log c + 3\right) \cdot \Pr\left(\sum_{i=1}^{14(\log^* m-1)} Z_i \geq (\log^* m - 1)\right).$$

Note that $\mathbf{E}\left(\sum_{j=1}^{14(2\log c+3)} Z_j\right) = 14(2\log c+3)/4$, and $\mathbf{E}\left(\sum_{i=1}^{14(\log^* m-1)} Z_i\right) = 14(\log^* m-1)/4$. Note further that for ξ_X, $m > c(2c+\log c+1)$, or equivalently, $\log^* m \geq 2$.

By Chernoff bounds [53, 56], for $\delta = 5/7$, we have:

$$\Pr\left(\sum_{j=1}^{14(2\log c+3)} Z_j < 2\log c + 3\right) < e^{-\frac{14(2\log c+3)(5/7)^2}{8}} \leq e^{-\frac{75}{28}},$$

and

$$\Pr\left(\sum_{i=1}^{14(\log^* m-1)} Z_i < (\log^* m - 1)\right) < e^{-\frac{14(\log^* m-1)(5/7)^2}{8}} \leq e^{-\frac{25}{28}}.$$

This follows that $\Pr(\xi_Y|\xi_X) \cdot \Pr(\xi_X) \geq \left(1 - e^{-\frac{75}{28}}\right)\left(1 - e^{-\frac{25}{28}}\right) \geq 1/2$. □

Corollary 4.7.13. *With probability at least* 0.99*, any deception DAG of size m shrinks to size zero in* $45(\log^* m + 2(\log c + 1))$ *rounds, where* $c = \frac{2(1+2p)}{\log e(1-2p)^2}$ *and* $p \leq \frac{1/2}{1+2\gamma}$*, for any constant* $\gamma > 0$*.*

Moreover, Table 4.1 shows that the more rounds *CHECK* has, the more probability of detecting corruptions is.

# Rounds	Probabilities
$15(\log^* m + 2(\log c + 1))$	0.6
$18(\log^* m + 2(\log c + 1))$	0.7
$21(\log^* m + 2(\log c + 1))$	0.8
$26(\log^* m + 2(\log c + 1))$	0.9

Table 4.1: # rounds versus probability of detecting corruptions

Now we show that for the adversary to maximize the number of rounds, in which no corruption detected, is to consider the maximum deception DAG in the first round.

Lemma 4.7.14. *For the adversary to maximize the expected number of rounds, in which no corruption detected, is to corrupt the output of the root party in the maximum deception DAG of the first round.*

Proof. For the case that the adversary considers multiple deception DAGs that are overlapped in the same round. Then the adversary corrupts more than one output in some round. Now let D' be the maximum deception DAG in this round. By Lemma 4.7.12, each of these deception DAGs shrinks to size zero in an expected number of rounds that is at most the expected number of rounds that D' shrinks to size zero.

Similarly, the case that the adversary considers multiple disjoint deception DAGs in the same round.

Therefore, for the adversary to maximize the expected number of rounds, in which no corruption detected, is to consider only the maximum deception DAG in the first round of *CHECK*. □

The next lemma shows that *CHECK* catches corruptions with probability $\geq 1/2$.

Lemma 4.7.15. *Assume some party selected uniformly at random in the last call to EVALUATE has corrupted a computation. Then when the algorithm CHECK is called, with probability at least* $1/2$*, some party will call RECOVER.*

Proof. Recall that the number of gates in the circuit is m. Thus, by Lemmas 4.7.12 and 4.7.14, this request is sent reliably to all input quorums in $O(\log^* m)$ rounds with probability at least $1/2$. Note that each request message, REQ^i, has a round number i. Hence, at any round, if any good party in any input quorum receives a request message of round number i and has not received $(i-1)$ request messages of proper round numbers, then it will call *RECOVER*.

If all input quorums receive all request messages properly in all $O(\log^* m)$ rounds, then *RECOMPUTE* must be called properly $O(\log^* m)$ times by all input quorums. By Lemmas 4.7.12 and 4.7.14, the result is computed and sent reliably to the output quorum in $O(\log^* m)$ rounds with probability at least 1/2.

Similarly, we know that in *RECOMPUTE*, the round number i is enclosed in REC^i, which is propagated with the computation results from the senders to the output quorum. Thus, at any round, if any good party in the output quorum receives a result with a round number i and has not received $(i-1)$ results with proper round numbers, then it will call *RECOVER*.

Finally, if all parties in the output quorum receive all results properly in all $O(\log^* m)$ rounds, then *RESEND* must be called $O(\log^* m)$ times by the output quorum. By Lemmas 4.7.12 and 4.7.14, the result of the computation is sent back reliably to all senders in $O(\log^* m)$ rounds with probability at least 1/2. Thus, the probability that *CHECK* succeeds in finding a corruption and calling *RECOVER* is at least 1/2. □

4.7.2 *RECOVER*

Lemma 4.7.16. *If some party selected uniformly at random in the last call to EVALUATE or CHECK has corrupted a computation, then RECOVER will identify a pair of neighboring quorums Q and Q' such that at least one pair of parties in these quorums is in conflict and at least one party in such pair is bad.*

Proof. First, we show that if a pair of parties x and y is in conflict, then at least one of them is bad. Assume not. Then both x and y are good. This implies that party x would have truthfully reported what it received and sent; any result that x has computed would have been sent directly to y; and y would have truthfully reported what it received from x. But this is a contradiction, since for x and y to be in conflict, y must have reported that it received from x something different than what x reported sending.

Now consider the case where a selected unmarked bad leader corrupted the computation in the last call to *EVALUATE*. By Lemma 4.7.15, with probability at least $1/2$, some party, $q' \in Q'$, will call *RECOVER*. Recall that in *RECOVER* q' broadcasts all messages it has received to all parties in Q'. These parties verify if q' received inconsistent messages before proceeding.

In *RECOVER*, we know that each party, $q \in Q$, participated in the last call to *COMPUTE* broadcasts what it has received and sent to all parties in Q. Thus, all parties of Q verify the correctness of q's computation. Thus, if the corruption occurs due to an incorrect computation made by a bad party, this corruption will be detected and all parties will know that this party is bad.

Now if all parties compute correctly and *CHECK* detects a corruption, then we show that there is some pair of parties will be in conflict. Assume this is not the case. Thus, by the definition of corruption, there must be a deception DAG, in which all inputs are provided correct and an output is corrupted at party q'. Then each pair of parties, $(q_j, q_k) \in (Q_j, Q_k)$, in the deception DAG that is rooted at q', is not in conflict, for $n+1 \leq j < k \leq m+n$. Thus, we have that 1) this DAG received all its inputs correct; 2) all parties compute correctly; and 3) no pair of parties is in conflict. This implies that it must be the case that q' received the correct output. But if this is the case, then q' that initially called *RECOVER* would have received no inconsistent messages. This is a contradiction since in such a case, this party would have been unsuccessful in trying to initiate a call to *RECOVER*. Thus, *RECOVER* will find two parties that are in conflict, and at least one of them will be bad. □

The next lemma bounds the number of calls to *RECOVER* before all bad parties are marked.

Lemma 4.7.17. *RECOVER is called $O(t)$ times before all bad parties are marked.*

Proof. By Lemma 4.7.16, if a corruption occurred in the last call to *EVALUATE*, and it is caught by *CHECK*, then *RECOVER* is called. *RECOVER* identifies at least one pair of parties that is in conflict, and each of such pairs has at least one bad party.

Now let b be the number of marked bad parties; and let g be the number of marked good parties. Also let

$$f(b,g) = b - \left(\frac{p}{1-p}\right) g.$$

Note that since $0 < p \leq \frac{1/2}{1+2\gamma}$, for any constant $\gamma > 0$, we have $0 < \frac{p}{1-p} \leq \frac{1}{1+4\gamma}$.

There are two cases at which $f(b,g)$ change.

Case 1: for each corruption caught, at least one bad party is marked. Thus, b increases by at least 1 and g increases by at most 1. This implies that $f(b,g)$ increases by at least $\left(\frac{1-2p}{1-p}\right)$.

Case 2: when $(1/2-\gamma)$-fraction of parties in any quorum Q get unmarked, for any constant $\gamma > 0$, b decreases by at most $p|Q|(1/2-\gamma)$ and g decreases by at least $(1-p)|Q|(1/2-\gamma)$. This implies that $f(b,g)$ further increases by at least 0.

Hence, $f(b,g)$ is monotonically increasing by at least $\left(\frac{1-2p}{1-p}\right)$ for each corruption caught. We know that when all bad parties are marked, $f(b,g) \leq t$.

Therefore, after at most $\left(\frac{1-p}{1-2p}\right)t$, or at most $\left(1+\frac{1}{2\gamma}\right)t$, calls to *RECOVER*, all bad parties are marked. □

4.7.3 Proof of Theorem 4.3.1

We first show the message cost, the number of operations and the latency of our algorithms. By Lemma 4.7.17, the number of calling *RECOVER* is at most $O(t)$. Thus, the resource cost of all calls to *RECOVER* is bounded as the number of calls to *COMPUTE* grows large.

Therefore, for the amortized cost, we consider only the cost of the calls to *EVALUATE* and *CHECK*.

When a computation is performed through a circuit of m gates with a circuit depth ℓ, *EVALUATE* has message cost $O(m + n\log n)$, number of operations $O(m + n\log n)$ and latency $O(\ell)$. *CHECK* has message cost $O((m + n\log n)(\log^* m)^2)$, number of operations $O((m+n\log n)\log^* m)$ and latency $O(\ell\log^* m)$, but *CHECK* is called only with probability $1/(\log^* m)^2$. Hence, the call to *CHECK* has an amortized expected message cost $O(m + n\log n)$, amortized computational operations $O(\frac{m+n\log n}{\log^* m})$ and an amortized expected latency $O(\ell/\log^* m)$.

In particular, if we call *COMPUTE* $\mathcal{L}$ times, then the expected total number of messages sent will be $O(\mathcal{L}(m + n\log n) + t(m\log^2 n))$ with expected total number of computational operations $O(\mathcal{L}(m+n\log n)+t(m\log n\log^* m))$ and latency $O(\ell(\mathcal{L}+t))$. This is true since *RECOVER* is called $O(t)$ times and each call to *RECOVER* has message cost $O(m\log^2 n)$ with computational operations $O(m\log n\log^* m)$ and latency $O(\ell)$.

Recall that by Lemma 4.7.17, the number of times *CHECK* must catch corruptions before all bad parties are marked is $O(t)$. In addition, if a bad party caused a corruption during a call to *EVALUATE*, then by Lemmas 4.7.15 and 4.7.16, with probability at least 1/2, *CHECK* will catch it. As a consequence, it will call *RECOVER*, which marks the parties that are in conflict. *RECOVER* is thus called with probability $1/(\log^* m)^2$, so the expected total number of corruptions is $O(t(\log^* m)^2)$.

4.8 Empirical Results

4.8.1 Setup

In this section, we empirically compare two algorithms via simulation. We measure the following resource costs of these algorithms: message cost, latency, probability of computation corruption and expected total number of corruptions. Note that the latency is measured as the number of message hops.

The first algorithm we simulate is *no-self-healing* (Section 4.4.3). This algorithm simply computes via all-to-all communication between quorums that are connected in binary-tree circuits. The second algorithm is *self-healing*, wherein we apply our self-healing algorithm in binary-tree circuits (Section 4.6).

In our experiments, we consider perfect binary-tree networks (in which each node has two children and all leaf nodes have the same depth) of sizes up to $n = 8{,}191$, where $\ell = \lfloor \log n \rfloor$ and the quorum size is $\lfloor 4 \log n \rfloor$. *COMPUTE* calls *CHECK* with probability $1/(\log^* n)^2$, and the subquorum size is $\lfloor 2 \log^* n \rfloor$. For multiparty computation, by Asharovy and Lindelly [6], we have that each call to MPC in a quorum of $O(\log n)$ nodes takes $O(\log^3 n)$ messages and $O(1)$ latency. Also, we do our experiments for several fractions of bad nodes such as f equal to 1/8, 1/16, 1/32 and 1/64, where $f = t/n$.

Our simulations consist of a sequence of calls to *COMPUTE* over the network. In each call to *COMPUTE*, we let the output quorum request the n input quorums to provide inputs in the same round to a circuit of m gates. Then the computation is performed through this circuit producing an output at the output quorum. This output is sent back to the input quorums.

We simulate an adversary who at the beginning of each simulation chooses uniformly at random without replacement a fixed number of nodes to control. Our adversary attempts to corrupt computations, where it drops messages or it changes the bits of messages sent between parties whenever possible. Aside from attempting to corrupt computations, the adversary performs no other attacks.

4.8.2 Results

The results of our experiments are shown in Figures 4.12, 4.13, 4.14, 4.15 and 4.16. Our results highlight two strengths of our self-healing algorithms (*self-healing*) when compared to algorithms without self-healing (*no-self-healing*). First, the message cost per *COMPUTE* decreases as the total number of calls to *COMPUTE* increases. Second, for a fixed number of calls to *COMPUTE*, the message cost per *COMPUTE* decreases as the total number of bad parties decreases. In particular, when there are no bad nodes, *self-healing* has dramatically less message cost than *no-self-healing*.

Expected Number of Messages

Figure 4.12 shows that before all bad parties are marked or all selected leaders are good: 1) the expected number of messages per call to *COMPUTE* decreases as the total number of calls to *COMPUTE* increases; and 2) for a fixed number of calls to *COMPUTE*, the expected number of messages per call to *COMPUTE* decreases as f decreases.

Table 4.2 shows that when all selected leaders are good, *self-healing* has dramatically less expected number of messages per call to *COMPUTE* than *no-self-healing*.

n	*self-healing*	*no-self-healing*
4,095	764,821	28,297,456
8,191	1,021,623	66,435,642

Table 4.2: Expected # messages per call to *COMPUTE* in *self-healing* and in *no-self-healing* for $n = 4{,}095$ and $n = 8{,}191$.

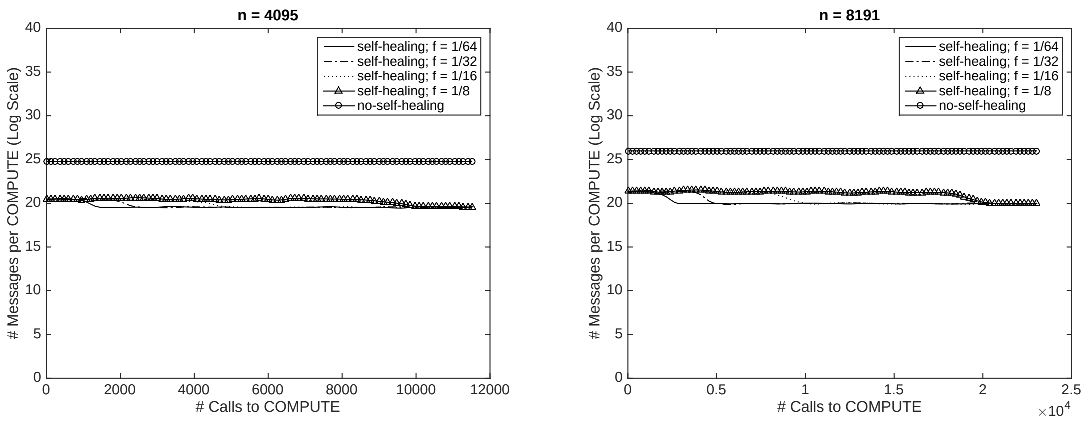

Figure 4.12: # messages per call to *COMPUTE* versus # calls to *COMPUTE*, for $n = 4{,}095$ and $n = 8{,}191$.

Improvement Factor of # Messages

Figure 4.13 shows the ratio of the expected number of messages of the *no-self-healing* algorithm to the expected number of messages of *COMPUTE*, after all bad nodes are marked or all selected leaders are good.

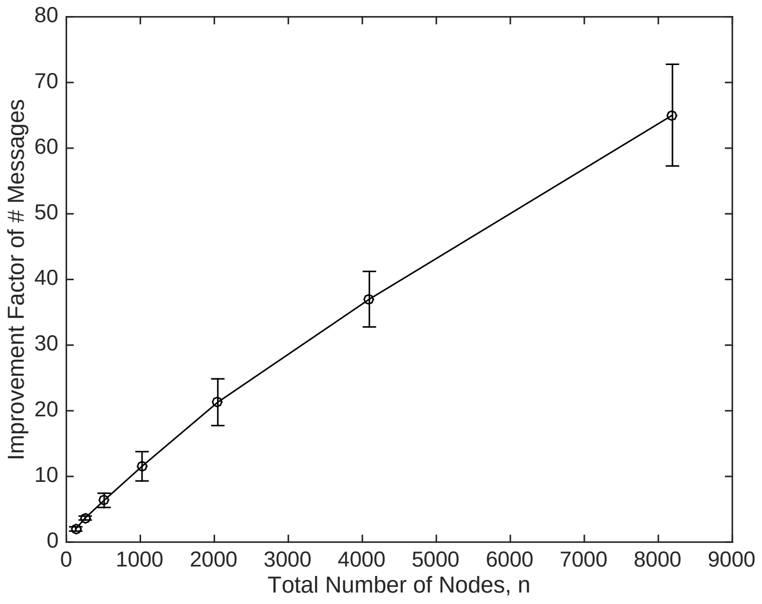

Figure 4.13: Improvement Factor of # Messages

Expected Latency

Figures 4.14 shows that the latency of *no-self-healing* is less than the latency of *self-healing* due to the expected latency of *CHECK*. Note that in any call to *CHECK*, if a corruption is detected at any round, then *CHECK* is terminated in this round and *RECOVER* is called. This figure shows that when all bad parties are marked (no more corruptions occur), each call to *CHECK* will run all $O(\log^* n)$ rounds.

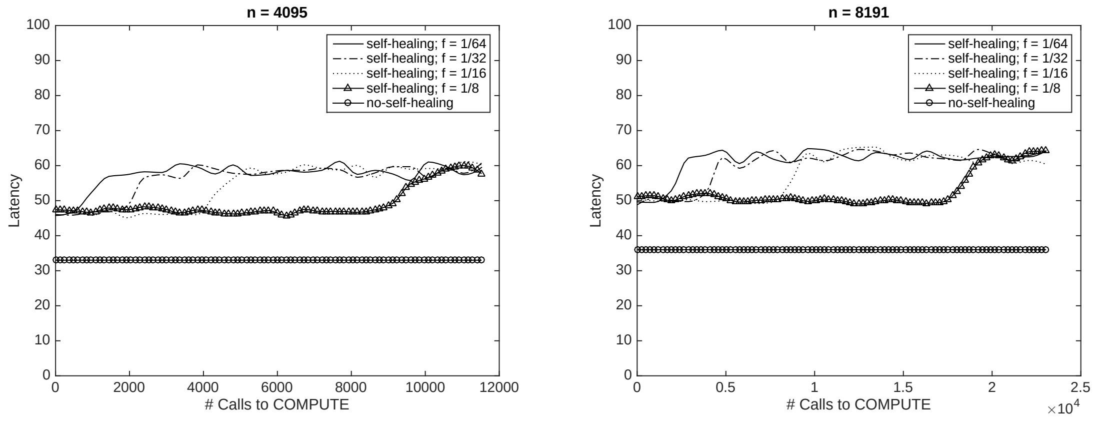

Figure 4.14: Latency per call to *COMPUTE* versus # calls to *COMPUTE*, for $n = 4{,}095$ and $n = 8{,}191$.

Table 4.3 shows that for $n = 4{,}095$ and $n = 8{,}191$, *self-healing* has latency that is at most twofold the latency of *no-self-healing*.

n	*self-healing*	*no-self-healing*
4,095	58	33
8,191	63	36

Table 4.3: Expected latency per call to *COMPUTE* in *self-healing* and in *no-self-healing* for $n = 4{,}095$ and $n = 8{,}191$.

Latency Increase Factor

Figure 4.15 shows the ratio of the expected latency of *COMPUTE* to the expected latency of the *no-self-healing* algorithm, after all bad nodes are marked or all selected leaders are good. It shows that the expected latency increases by a factor of $\Theta(1)$.

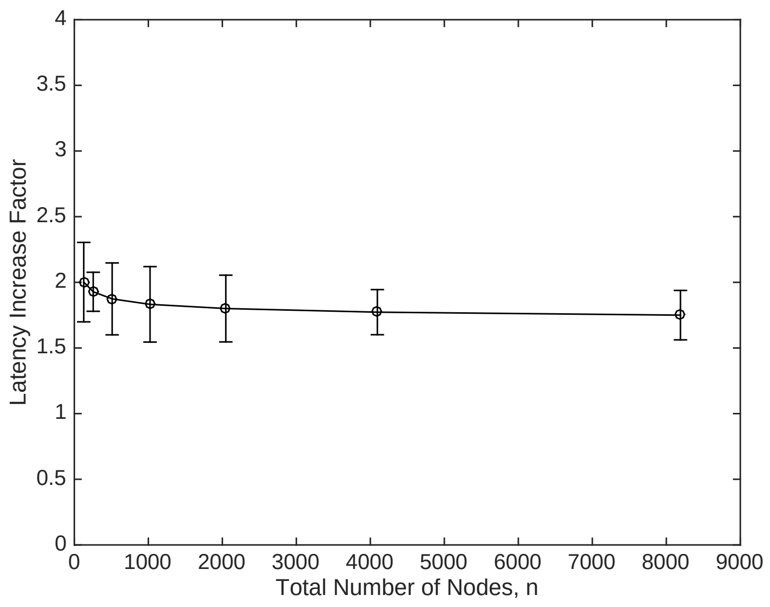

Figure 4.15: Latency Increase Factor

Probability of Computation Corrupted

Figure 4.16 shows that the probability of computation corrupted per a call to *COMPUTE* decreases as the total number of calls to *COMPUTE* increases. Also, for a fixed number of calls to *COMPUTE*, this probability decreases as the total number of bad nodes decreases.

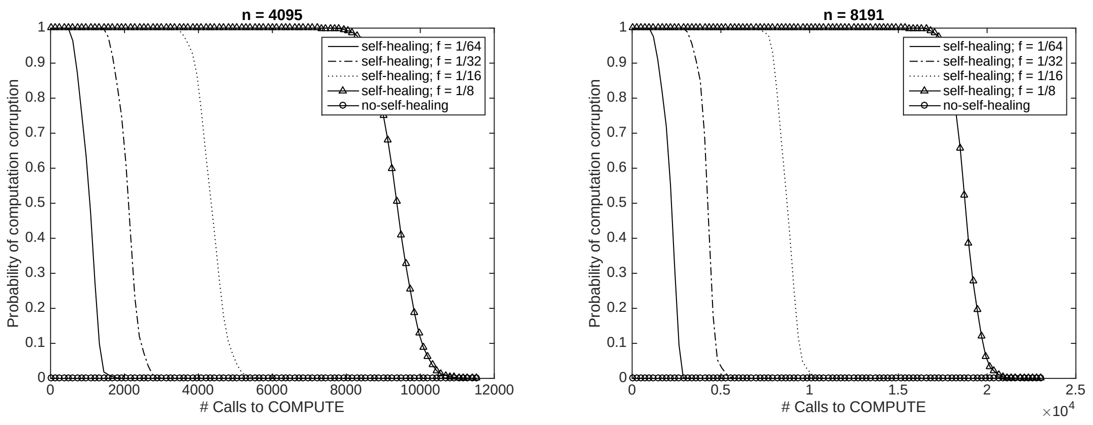

Figure 4.16: The probability that the computation is corrupted versus # calls to *COMPUTE*, for $n = 4{,}095$ and $n = 8{,}191$.

Expected Total Number of Corruptions

In Figure 4.16, for each network given the number of parties and the fraction of bad parties, if we integrate the corresponding curve, then we get $\Sigma(n)$, which is the expected total number of times that the computation is corrupted in all calls to *COMPUTE* in a network of n parties.

Table 4.4 shows the fact that $\Sigma(n) = O(t(\log^* n)^2)$. In particular, $\Sigma(n) \leq \sigma(n)$, where

$$\sigma(n) = 2\left(\frac{1-2f}{1-4f}\right) t(\log^* n)^2.$$

f	$\Sigma(4{,}095)$	$\sigma(4{,}095)$	$\Sigma(8{,}191)$	$\sigma(8{,}191)$
1/64	1,029	2,116	2,069	4,232
1/32	2,097	4,388	4,214	8,777
1/16	4,351	9,557	8,720	19,114
1/8	9,366	24,576	18,759	49,152

Table 4.4: Total # corruptions for $n = 4{,}095$ and $n = 8{,}191$.

Chapter 5

Conclusion and Future Work

"The empires of the future are the empires of the mind."

– Winston Churchill

5.1 Selfishness

5.1.1 Results

In this dissertation, we studied a variant of the El Farol game that had both positive and negative network effects. Our goal was to maximize social welfare.

We know that selfishness of players deteriorates social welfare. Our goal is to maximize the social welfare of our extended El Farol game. The basic idea of our approach is to design an optimal mediator, which implements the best correlated equilibrium for this game. Note that a mediator is an external trusted entity that advises each player separately and privately with an action convincing the player that no incentive to deviate from the given advice. Note further that optimal mediator is a mediator that maximizes the social welfare over all mediators.

In this dissertation, we have provided a full characterization of the best correlated equilibrium of the El Farol game. In particular, we have shown that the best correlated equilibrium has two strategy profiles. Further, for certain values of c, s_1 and s_2, the probability distribution on these two strategy profiles enables the optimal mediator to almost surely select one strategy profile, which is the best Nash equilibrium.

Moreover, we characterize the value of correlation, which shows the efficiency of mediation subject to the best Nash equilibrium and the optimal social cost of benevolent players. The value of correlation has two metrics: the mediation value and the enforcement value. The mediation value is the ratio of the social cost of our optimal mediator to the social cost of the best Nash equilibrium; and the enforcement value is the ratio of the social cost of our optimal mediator to the optimal social welfare of benevolent players.

We show that the mediation value in our game can be unbounded in the sense that using our mediator significantly improves the social cost of our game compared to the best Nash equilibrium. Also, we show that the enforcement value can be unbounded, where our optimal mediator is inefficient compared to the optimal social cost of benevolent players.

5.1.2 Open Problems

Several open questions remain. First, can we generalize our results for a game where the players choose among more than two actions? Formally, we consider the following problem. The Multi-Site El Farol game is a non-cooperative game, in which there are multiple sites and each player must choose one site. All sites have identical cost functions of both types of network effects.

Formally, given a set of n selfish players, and a set of k sites. Each player has to select one site to go to, from the k sites. Each site has three parameters c, s_1 and s_2, where $0 < c < s_1$ and $s_2 > 0$. If x is the fraction of players to go to site i, then the cost $f_i(x)$ for any player that goes to site i is as follows:

$$f_i(x) = \begin{cases} c - s_1 x & 0 \leq x \leq \frac{c}{s_1}, \\ s_2(x - \frac{c}{s_1}) & \frac{c}{s_1} \leq x \leq 1. \end{cases}$$

We assume that all sites have identical cost function, so we omit index i from $f_i(x)$. We further assume that this game is non-atomic and symmetric. The objective is to measure the value of correlation (MV and EV) for this game. To measure these values, we consider the following. What is the optimal solution when the players are benevolent? What is the best Nash equilibrium? What is the best correlated equilibrium? Moreover, it is interesting to determine how MV and EV change for fixed c, s_1 and s_2, as the number of sites changes.

A second interesting problem is as follows. We assume in our analysis that the utility function is composed of linear negative and positive network effects. However, if the utility function, $f(x)$, is any polynomial of the fraction of players that go, what is the characterization of optimal mediator? Is the minimum number of required strategy profiles of an optimal mediator strongly related to the degree of this polynomial?

A third interesting problem is to maximize the social welfare if: a) our game becomes an atomic game in the sense that at least one player has a significant influence on the outcome; b) our game will have asymmetric players in the sense that at least two players have two different utility functions; or c) our game becomes a cooperative game in the sense that the players can cooperate having coalitions?

5.2 Malice

5.2.1 Results

There are two common fault models: fail-stop faults and Byzantine faults. In the fail stop model, once a node in the system fails, it stops functioning and does not produce an output.

Byzantine faults are more challenging. An adversary is assumed to take over a subset of nodes causing Byzantine faults, where these nodes behave maliciously and may collude in order to disrupt the system.

Replication and self-healing are two common approaches to achieve fault tolerance. In replication, multiple nodes perform the same function redundantly. Unfortunately, replication requires expensive communication complexity and hardware cost even if the system is not under attack. In self-healing algorithms, the network detects the corruptions, investigates the failures and recovers automatically. These algorithms detect corruptions efficiently and expend additional resources only when necessary in order to recover from faults.

Many papers have been conducted to self-heal networks in the fail-stop model. To the best of our knowledge, there is no self-healing algorithm developed for recovering from Byzantine attacks.

In this thesis, we developed self-healing algorithms that detect Byzantine corruptions and quarantine the Byzantine nodes. We designed such algorithms for both communication and computation networks. Our algorithms reduce message cost significantly compared to algorithms that are not self-healing. The price we pay for this improvement is the possibility of corruption.

We have proven two interesting lemmas for corruption detection. In these lemmas, we have shown that there exists an efficient algorithm that checks if there was a corruption. This algorithm proceeds in rounds. In each round, the longest substring (or the largest subtree) of Byzantine nodes that were involved in the corruption is expected to shrink logarithmically. The first lemma is for communication networks. This lemma is trivial where if you flip a coin x times independently, and the coin is tail with constant probability, then the probability of having a substring of tails of size $\Omega(\log x)$ is less than $1/2$. The second lemma is for computation networks. It shows that given any tree of size n, in which each node survives independently with a constant probability, then the probability of having a subtree, of surviving nodes of size $\Omega(\log n)$, is less than 1/2. We also extend this result for circuits.

When a corruption is detected, a recovery procedure is triggered in order to quarantine the bad nodes. In this procedure, each node announces all messages that it has received and sent. Then we mark any conflicting pair of nodes; note that any two nodes are in conflict if they claim differently in terms of sending and receiving the messages.

5.2.2 Open Problems

Several open questions remain. In our self-healing algorithms, the expected total number of corruptions is $O(t(\log^* n)^2))$, where t is the number of bad nodes and n is the number of nodes. This is slightly higher than a linear function of the total number of bad nodes. Can we reduce the number of corruptions to $O(t)$ keeping the same message cost and latency? Also, we assume a partially synchronous communication model, which is crucial for our *CHECK* algorithm to detect corruptions over rounds. Can we extend this algorithm to work in asynchronous networks? Additionally, we assume a static adversary which must choose which nodes to control at the beginning of our algorithm. Can we modify our algorithms to self-heal networks in the presence of adaptive adversary, which can take over nodes at any point during the course of the protocol? Also, can we deal with churn?

In communication networks, we assumed that the sender is a good node. Can we extend our algorithms to tolerate the case where the sender is bad in such a way that 1) our algorithms will remain efficient in terms of message cost; and 2) the network is resistant to denial of service attacks?

Moreover, in each round of our *CHECK* algorithm, the sender provides an array of random integers to select the nodes that will participate in this round uniformly at random. Note that each array has $O(\ell \log n \log\log n)$ bits. If the message m has b bits, then the communication complexity per call to *SEND* is $O((\ell + \log n)(\ell \log n \log\log n + b))$, and for naive communication, it is $O(\ell \log^2 n \cdot b)$. In order to improve the communication complexity of our algorithms, can we reduce the number of bits that is required to represent such arrays to $O(\ell \log\log n)$?

Finally, in computation networks, we do not protect the privacy of the inputs of the parties. Can we modify our algorithms to maintain the privacy of these inputs, i.e., implementing secure multiparty computation [6, 16, 17, 68, 83]? Also, in a circuit of n inputs and m gates, each array of random integers produced in *CHECK* has $O(m \log n \log\log n)$ bits. Assuming that any input message has $O(b)$ bits, then the communication complexity per call to *COMPUTE* is $O((m + n \log n)(m \log n \log\log n + b))$, and $O((m + n) \log^2 n \cdot b)$ for naive computation. Can we reduce the size of such an array to $O(m \log\log n)$?

Decentralized Approach

Now we suggest a decentralized approach that may: 1) reduce the communication complexity of the arrays of random integers in *CHECK*; and 2) tolerate the possibility of having a bad sender.

First, in order to reduce the communication complexity of these arrays to $O(\log\log n)$, we construct static paths in the sense that we initially select a set of $O(\log n)$ paths. Each path is a vector of nodes; one node from each quorum in the quorum path is selected uniformly at random, after the static adversary takes over the Byzantine nodes. Second, we modify our *CHECK* algorithm to be decentralized in order to tolerate the case that the sender is bad. In particular, for each round, one path in expectation is selected uniformly at random from the set of static paths, in a decentralized manner. This path is added to the paths that have been previously selected in this call to *CHECK*. Then the nodes in the leftmost quorum resend the message to the rightmost quorum through the nodes of all added paths, via all-to-all communication.

Now we show how a path is selected u.a.r. in a decentralized manner. We let each node

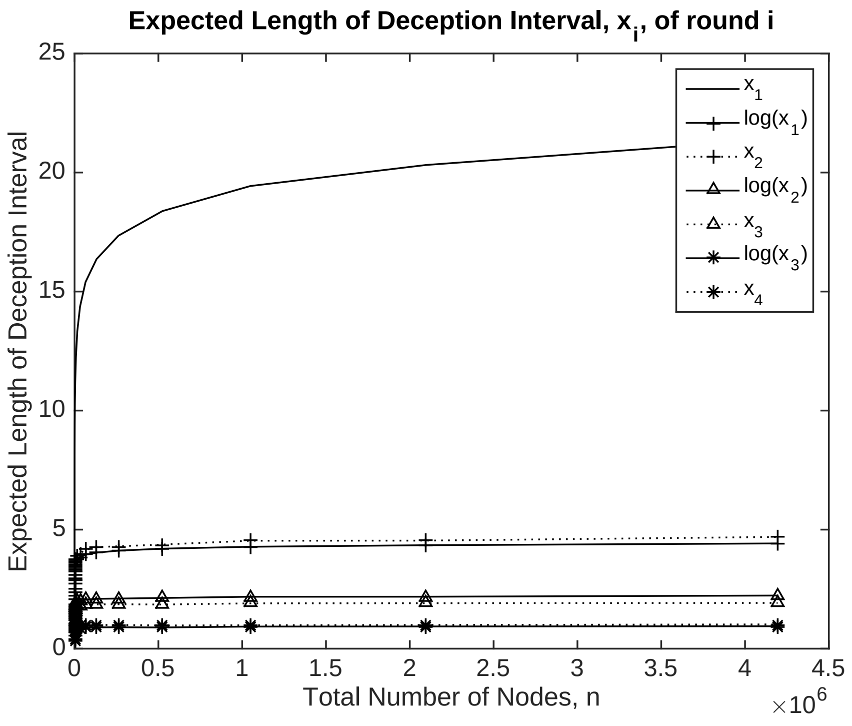

Figure 5.1: Deception Intervals in Decentralized Approach

wake up independently with some probability so that one node in expectation wakes up. Once a node wakes up, it does the following: 1) it notifies all nodes in its quorum that a new path (i.e., the path that this node is in) is activating; and 2) it asks all nodes in its quorum for the paths that have been previously activated in this call to *CHECK*. Then all nodes, in the current path and the paths that have been activated, send a message (containing the ids of these paths) to the sender and the receiver via all-to-all communication.

A challenge in this algorithm is the fact that for every round, when a node wakes up and the message is supposed to propagate via all-to-all communication to all good nodes in the selected static paths, there is no guarantee that all good nodes in these paths will receive this message due to the presence of the Byzantine nodes. This affects the probability that the deception interval shrinks logarithmically from round to round. If it is possible to show that this probability remains constant, then the expected number of rounds required to detect corruptions is still $O(\log^* n)$.

We run a simulation of this algorithm. In this simulation, we have a quorum path of length n and a quorum size of $\log n$, for $2^4 \leq n \leq 2^{22}$. Figure 5.1 plots the average length of the longest deception intervals for several consecutive rounds. This simulation suggests that the deception intervals shrink logarithmically, in expectation, from round to round.

Recall that when a corruption is detected, the recovery procedure is triggered to identify at least two nodes that are in conflict. Note that any two nodes, that are in conflict in the static paths, are replaced with two other nodes. This implies that these static paths are tuned with the new selected nodes.

For computation networks, in order to reduce the communication complexity of *CHECK*, we may extend the decentralized approach we suggest for communication networks. In particular, we initially select $O(\log n)$ static circuits after the adversary has chosen the Byzantine nodes. Each circuit is a directed acyclic graph of nodes, and each node is selected u.a.r. from a quorum that represents a gate in the computation network. In each round, one node in expectation wakes up and initiates a checking of the computation of the circuit that the node is in. If a corruption is detected, then a recovery procedure is triggered to quarantine the bad nodes and to tune the static circuits accordingly.

5.3 Final Conclusion

In game theory, selfishness impairs social welfare compared to the optimal social welfare of benevolent players. Many social and economic situations have both positive and negative network effects. Our objective was to find an equilibrium that maximizes the social welfare for these situations. Unfortunately, the best Nash equilibrium may have a poor social welfare. Correlated equilibrium is a generalization of the Nash equilibrium. Finding the best correlated equilibrium was an objective to show how much it can ameliorate the social welfare.

In this dissertation, we have shown that the best correlated equilibrium can substantially improve the social welfare compared to the best Nash equilibrium in a game that has both positive and negative network effects. We have provided an exact characterization of the best correlated equilibrium, and we have described how efficient the best correlated equilibrium is, compared to the best Nash equilibrium and the optimal social welfare. Interestingly, there are certain cases in which the best correlated can infinitely improve the social welfare compared to the best Nash equilibrium.

Fault tolerance is an essential requirement for enormous large-scale distributed systems. Replication and self-healing are two common approaches to attain fault tolerance. Self-healing has advantages over replication, where self-healing has less communication complexity and it spends resources only when necessary. Many papers have been conducted to self-heal networks in the presence of fail-stop faults. To the best of our knowledge, there is no self-healing algorithm that recovers networks from Byzantine faults.

We have provided self-healing algorithms to recover communication and computation networks from Byzantine faults. Our self-healing algorithms can efficiently detect corruptions and quarantine Byzantine nodes. We have shown that each time a corruption is detected, there is a progress towards quarantining all Byzantine nodes, after which no more corruptions occur.

Finally, we have described several open problems for selfishness and malice. Moreover, we have proposed a decentralized approach, for future work, that may improve the communication complexity of our self-healing algorithms and let our algorithms work in a fully decentralized manner.

References

[1] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, PODC'06, pages 53–62, New York, NY, USA, 2006. ACM.

[2] I. Abraham, D. Dolev, and J. Y. Halpern. Lower bounds on implementing robust and resilient mediators. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 302–319, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[4] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proceedings of 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'04, pages 295–304, 2004.

[5] B. Arthur. Bounded rationality and inductive behavior (the El Farol problem). *American Economic Review*, 84:406–411, 1994.

[6] G. Asharov and Y. Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. *Electronic Colloquium on Computational Complexity*, 18:36, 2011.

[7] I. Ashlagi, D. Monderer, and M. Tennenholtz. Mediators in position auctions. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, EC'07, pages 279–287, New York, NY, USA, 2007. ACM.

[8] I. Ashlagi, D. Monderer, and M. Tennenholtz. On the value of correlation. *Journal of Artificial Intelligence Research (JAIR)*, 33:575–613, 2008.

[9] J. Aspnes, K. Chang, and A. Yampolskiy. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. *Journal of Computer and System Science*, 72(6):1077–1093, 2006.

[10] R. J. Aumann. Subjectivity and correlation in randomized games. *Mathematical Economics*, 1:67–96, 1974.

[11] R. J. Aumann and L.S. Shapley. *Values of Non-Atomic Games.* A Rand Corporation Research Study Series. Princeton University Press, 1974.

[12] B. Awerbuch and C. Scheideler. Towards scalable and robust overlay networks. In *the 6th International workshop on Peer-To-Peer Systems*, IPTPS'07, 2007.

[13] B. Awerbuch and C. Scheideler. Robust random number generation for peer-to-peer systems. volume 410, pages 453–466, Essex, UK, February 2009. Elsevier Science Publishers Ltd.

[14] B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. *Theory of Computing Systems*, 45(2):234–260, 2009.

[15] V. S. Bawa. Optimal rules for ordering uncertain prospects. *Journal of Financial Economics*, 2(1):95–121, 1975.

[16] D. Beaver. Efficient multiparty protocols using circuit randomization. CRYPTO91, pages 420–432. 1992.

[17] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[18] C. G. Bird. On the stability of a class of noncooperative games. *Linear Algebra and its Applications*, 40(0):119 – 127, 1981.

[19] I. Boman, J. Saia, C. Abdallah, and E. Schamiloglu. Brief announcement: Self-healing algorithms for reconfigurable networks. In *Stabilization, Safety, and Security of Distributed Systems*, volume 4280 of *SSS'06*, pages 563–565, 2006.

[20] M. Bradonjic, G. Ercal-Ozkaya, A. Meyerson, and A. Roytman. On the price of mediation. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, EC'09, pages 315–324, New York, NY, USA, 2009. ACM.

[21] F. Brandt, F. Fischer, P. Harrenstein, and Y. Shoham. A game-theoretic analysis of strictly competitive multiagent scenarios. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1199–1206, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[22] F. Brandt, F. Fischer, and Y. Shoham. On strictly competitive multi-player games. In *Proceedings of the 21st national conference on Artificial intelligence*, volume 1 of *AAAI'06*, pages 605–612. AAAI Press, 2006.

[23] D. Challet, M. Marsili, and G. Ottino. Shedding light on El Farol. *Physica A: Statistical Mechanics and Its Applications*, 332:469–482, 2004.

[24] G. Christodoulou and E. Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *Proceedings of the 13th annual European Symposium on Algorithms*, ESA'05, pages 59–70, Berlin, Heidelberg, 2005. Springer-Verlag.

[25] M. Datar. Butterflies and peer-to-peer networks. In *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *ESA'02*, pages 310–322, 2002.

[26] M. A. R. De Cara, O. Pla, and F. Guinea. Competition, efficiency and collective behavior in the "El Farol" bar model. *The European Physical Journal B - Condensed Matter and Complex Systems*, 10(1):187–191, 1999.

[27] J. Díaz, D. Mitsche, N. Rustagi, and J. Saia. On the power of mediators. In *Proceedings of the 5th International Workshop on Internet and Network Economics*, WINE '09, pages 455–462, Berlin, Heidelberg, 2009. Springer-Verlag.

[28] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, New York, NY, USA, 2010.

[29] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'02, pages 94–103, 2002.

[30] A. Fiat and J. Saia. Censorship resistant peer-to-peer networks. *Theory of Computing*, 3(1):1–23, 2007.

[31] A. Fiat, J. Saia, and M. Young. Making chord robust to Byzantine attacks. In *In Proceedings of the 13th Annual European Symposium on Algorithms*, ESA'05, pages 803–814, 2005.

[32] F. Forges. An approach to communication equilibria. *Econometrica*, 54(6):1375–1385, 1986.

[33] F. Forgó. A generalization of correlated equilibrium: A new protocol. *Mathematical Social Sciences*, 60(3):186–190, 2010.

[34] F. Forgó. Measuring the power of soft correlated equilibrium in 2-facility simple non-increasing linear congestion games. *Central European Journal of Operations Research*, pages 1–17, 2012.

[35] T. Frisanco. Optimal spare capacity design for various protection switching methods in ATM networks. In *IEEE International Conference on Communications*, volume 1 of *ICC'97*, pages 293–298, 1997.

[36] J. Hadar and W. R. Russell. Rules for Ordering Uncertain Prospects. *American Economic Review*, 59(1):25–34, 1969.

[37] T. P. Hayes, N. Rustagi, J. Saia, and A. Trehan. The forgiving tree: a self-healing distributed data structure. PODC'08, pages 203–212, 2008.

[38] T. P. Hayes, J. Saia, and A. Trehan. The forgiving graph: a distributed data structure for low stretch under adversarial attack. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC'09, pages 121–130, 2009.

[39] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of the 17th International Symposium on Distributed Computing*, volume 2848 of *DISC '03*, pages 321–336, 2003.

[40] R. R. Iraschko, M. H. MacGregor, and W. D. Grover. Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks. *IEEE/ACM Transactions on Networking*, 6(3):325–336, 1998.

[41] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs.* Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

[42] A. Xin Jiang, A. D. Procaccia, Y. Qian, N. Shah, and M. Tambe. Defender (mis)coordination in security games. In *International Joint Conference on Artificial Intelligence*, IJCAI'13, 2013.

[43] A. Kapadia and N. Triandopoulos. Halo: High-assurance locate for distributed hash tables. In *the 15th Annual Network and Distributed System Security Symposium*, NDSS'08, 2008.

[44] R. M. Karp. The transitive closure of a random digraph. *Random Structures & Algorithms*, 1(1):73–93, 1990.

[45] A. Kate and I. Goldberg. Distributed key generation for the internet. In *29th IEEE International Conference on Distributed Computing Systems*, ICDCS'09, pages 119–128, 2009.

[46] V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable Byzantine agreement through quorum building, with full information. In *12th International Conference on Distributed Computing and Networking*, ICDCN'11, pages 203–214, 2011.

[47] A. Klenke and L. Mattner. Stochastic ordering of classical discrete distributions. *Advances in Applied Probability*, 42(2):392 – 410, 2010.

[48] J. Knockel, G. Saad, and J. Saia. Self-healing of Byzantine faults. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS'13, pages 98–112, 2013.

[49] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, STACS'99, pages 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[50] H. Lus, C. Aydin, S. Keten, H. Unsal, and A. Atiligan. El Farol revisited. *Physica A: Statistical Mechanics and Its Applications*, 346(3-4):651–656, 2005.

[51] A. Martin-Lf. Symmetric sampling procedures, general epidemic processes and their threshold limit theorems. *Journal of Applied Probability*, 23(2):pp. 265–282, 1986.

[52] D. Mitsche, G. Saad, and J. Saia. The power of mediation in an extended El Farol game. In Berthold Vcking, editor, *Algorithmic Game Theory*, volume 8146 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin Heidelberg, 2013.

[53] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[54] D. Monderer and M. Tennenholtz. Strong mediated equilibrium. *Artificial Intelligence*, 173(1):180–195, 2009.

[55] T. Moscibroda, S. Schmid, and R. Wattenhofer. When selfish meets evil: Byzantine players in a virus inoculation game. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of Distributed Computing*, PODC'06, pages 35–44, New York, NY, USA, 2006. ACM.

[56] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[57] K. Murakami and H. S. Kim. Comparative study on restoration schemes of survivable ATM networks. In *Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1 of *INFOCOM'97*, pages 345–352, 1997.

[58] R. B. Myerson. Multistage games with communication. *Econometrica*, 54(2):323–58, March 1986.

[59] M. Naor and U. Wieder. Novel architectures for p2p applications: The continuous-discrete approach. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 50–59, New York, NY, USA, 2003. ACM.

[60] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *2nd International Workshop on Peer-to-Peer Systems*, IPTPS'03, pages 88–97, 2003.

[61] J. F. Nash. Equilibrium points in n-person games. *Proc. of the National Academy of Sciences*, 36:48–49, 1950.

[62] J. F. Nash. *Non-cooperative Games*. Princeton University, 1950.

[63] R. O'Donnell. *CS 15-359: Probability and Computing (Lecture Notes)*. Carnegie Mellon University, 2009. `http://www.cs.cmu.edu/~odonnell/papers/probability-and-computing-lecture-notes.pdf`.

[64] G. Pandurangan and A. Trehan. Xheal: localized self-healing using expanders. In *30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC'11, pages 301–310, 2011.

[65] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *J. ACM*, 55(3):1–29, August 2008.

[66] B. Peleg and A. D. Procaccia. Implementation by mediated equilibrium. *International Journal of Game Theory*, 39(1-2):191–207, 2010.

[67] M. Prabhakaran and A. Sahai. *Secure Multi-Party Computation*, volume 10. IOS Press, 2013.

[68] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, STOC'89, pages 73–85, 1989.

[69] O. Rozenfeld and M. Tennenholtz. Strong and correlated strong equilibria in monotone congestion games. In *Proceedings of the 2nd International Workshop on Internet and Network Economics*, WINE'06, pages 74–86, Berlin, Heidelberg, 2006. Springer-Verlag.

[70] O. Rozenfeld and M. Tennenholtz. Group dominant strategies. In *Proceedings of the 3rd International Workshop on Internet and Network Economics*, WINE'07, pages 457–468, Berlin, Heidelberg, 2007. Springer-Verlag.

[71] O. Rozenfeld and M. Tennenholtz. Routing mediators. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, IJCAI'07, pages 1488–1493, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[72] G. Saad and J. Saia. Self-healing computation. In *16th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, SSS'14, pages 195–210, 2014.

[73] J. Saia and A. Trehan. Picking up the pieces: Self-healing in reconfigurable networks. In *IEEE International Parallel and Distributed Processing Symposium*, IPDPS'08, pages 1–12, 2008.

[74] J. Saia and M. Young. Reducing communication costs in robust peer-to-peer networks. *Information Processing Letters*, 106(4):152–158, 2008.

[75] A. D. Sarma and A. Trehan. Edge-preserving self-healing: keeping network backbones densely connected. In *IEEE Conference on Computer Communications Workshops*, INFOCOM WKSHPS, pages 226–231, 2012.

[76] C. Scheideler. How to spread adversarial nodes? rotate! In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, STOC'05, pages 704–713, 2005.

[77] D. Schmeidler. Equilibrium points of nonatomic games. *Journal of Statistical Physics*, 7(4):295–300, 1973.

[78] M. Shaked and J.G. Shanthikumar. *Stochastic orders and their applications*. Probability and mathematical statistics. Academic Press, 1994.

[79] M. Tennenholtz. Game-theoretic recommendations: some progress in an uphill battle. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '08, pages 10–16, Richland, SC, 2008.

[80] J. R. Vacca. *Public Key Infrastructure: Building Trusted Applications and Web Services*. CRC Press, 2004.

[81] B. Van Caenegem, N. Wauters, and P. Demeester. Spare capacity assignment for different restoration strategies in mesh survivable networks. In *IEEE International Conference on Communications*, volume 1 of *ICC '97*, pages 288–292, 1997.

[82] Y. Xiong and L. G. Mason. Restoration strategies and spare capacity requirements in self-healing ATM networks. *IEEE/ACM Transactions on Networking*, 7(1):98–110, 1999.

[83] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS'82, pages 160–164, 1982.

[84] M. Young, A. Kate, I. Goldberg, and M. Karsten. Practical robust communication in DHTs tolerating a Byzantine adversary. In *The 30th International Conference on Distributed Computing Systems*, ICDCS'10, pages 263–272, 2010.

[85] M. Young, A. Kate, I. Goldberg, and M. Karsten. Towards practical communication in Byzantine-resistant DHTs. *IEEE/ACM Transactions on Networking*, 21(1):190 –203, 2013.