

Resource-Efficient Communication in the Presence of Adversaries

by

Maxwell Young

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2011

© Maxwell Young 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This dissertation presents algorithms for achieving communication in the presence of adversarial attacks in large, decentralized, resource-constrained networks. We consider abstract *single-hop* communication settings where a set of senders \mathcal{S} wishes to directly communicate with a set of receivers \mathcal{R} . These results are then extended to provide resource-efficient, multi-hop communication in wireless sensor networks (WSNs), where energy is critically scarce, and peer-to-peer (P2P) networks, where bandwidth and computational power are limited. Our algorithms are provably correct in the face of attacks by a computationally bounded adversary who seeks to disrupt communication between correct participants.

The first major result in this dissertation addresses a general scenario involving single-hop communication in a time-slotted network where a single sender in \mathcal{S} wishes to transmit a message m to a single receiver in \mathcal{R} . The two players share a communication channel; however, there exists an adversary who aims to prevent the transmission of m by periodically blocking this channel. There are costs to send, receive or block m on the channel, and we ask: How much do the two players need to spend relative to the adversary in order to guarantee transmission of the message?

This problem abstracts many types of conflict in information networks, and the associated costs represent an expenditure of network resources. We show that it is significantly more costly for the adversary to block m than for the two players to achieve communication. Specifically, if the cost to send, receive and block m in a slot are fixed constants, and the adversary spends a total of B slots to try to block the message, then both the sender and receiver must be active in only $O(B^\varphi + 1) = O(B^{.62} + 1)$ slots in expectation to transmit m , where $\varphi = (1 + \sqrt{5})/2$ is the golden ratio. Surprisingly, this result holds even if (1) the value of B is *unknown* to either player; (2) the adversary knows the algorithms of both players, but not their random bits; and (3) the adversary is able to launch attacks using total knowledge of past actions of both players. Finally, these results are applied to two concrete problems. First, we consider jamming attacks in WSNs and address the fundamental task of propagating m from a single device to all others in a WSN in the presence of faults; this is the problem of *reliable broadcast*. Second, we examine how our algorithms can mitigate application-level distributed denial-of-service attacks in wired client-server scenarios.

The second major result deals with a single-hop communication problem where now \mathcal{S} consists of multiple senders and there is still a single receiver who wishes to obtain a message m . However, many of the senders (strictly less than half) can be faulty, failing to send m or sending incorrect messages. While the majority of the senders possess m , rather than listening to all of \mathcal{S} and majority filtering on the received data, we desire an algorithm that allows the single receiver to decide on m in a more efficient manner. To investigate this scenario, we define and devise algorithms for a new data streaming problem called the *Bad Santa problem* which models the selection dilemma faced by the receiver.

With our results for the Bad Santa problem, we consider the problem of energy-efficient reliable broadcast. All previous results on reliable broadcast require devices to spend significant time in the energy-expensive receiving state which is a critical problem in WSNs where devices are typically battery powered. In a popular WSN model, we give a reliable broadcast protocol that achieves optimal fault tolerance (i.e., tolerates the maximum number of faults in this WSN model) and improves over previous results by achieving an expected quadratic decrease in the cost to each device. For the case where the number of faults is within a $(1 - \epsilon)$ -factor of the optimal fault tolerance, for any constant $\epsilon > 0$, we give a reliable broadcast protocol that improves further by achieving an expected (roughly) exponential decrease in the cost to each device.

The third and final major result of this dissertation addresses single-hop communication where \mathcal{S} and \mathcal{R} both consist of multiple peers that need to communicate in an attack-resistant P2P network. There are several analytical results on P2P networks that can tolerate an adversary who controls a large number of peers and uses them to disrupt network functionality. Unfortunately, in such systems, operations such as data retrieval and message sending incur significant communication costs. Here, we employ cryptographic techniques to define two protocols both of which are more efficient than existing solutions. For a network of n peers, our first protocol is *deterministic* with $O(\log^2 n)$ message complexity and our second protocol is randomized with expected $O(\log n)$ message complexity; both improve over all previous results. The hidden constants and setup costs for our protocols are small and no trusted third party is required. Finally, we present an analysis showing that our protocols are practical for deployment under significant churn and adversarial behaviour.

Acknowledgements

I would like to thank my supervisors Martin Karsten and Raouf Boutaba. Despite being fairly set in my ways, Martin’s unerring ability to rapidly uncover the core ideas of a problem has reshaped my approach to research and I feel privileged to have worked with him. Raouf, in addition to providing critical guidance, has been perpetually supportive of my endeavors and provided many opportunities over the years that greatly facilitated my growth as a researcher. Finally, I am indebted to Jared Saia who taught me the ropes as a young researcher several years ago and who continues to be both an “unofficial” supervisor and a friend.

I am grateful to the other members of my committee Christian Scheideler, Ian Goldberg, Alex Lopez-Ortiz, and Paul Ward. Their extensive feedback has greatly improved this thesis and elevated it above a simple regurgitation of published papers.

There are a number of researchers that I wish to thank, both with regards to the contents of this thesis and other projects completed during my studies. It has been a pleasure to work with Valerie King, Cynthia Phillips, and Jared Saia on several problems from the area of wireless sensor networks. I am also fortunate to have collaborated with Aniket Kate, Ian Goldberg, and Martin Karsten on the topic of Byzantine fault tolerance in peer-to-peer networks. In the area of radiation therapy, I am lucky to have worked with Therese Biedl, Stephane Durocher, Samuel Fiorini, and Holger Hoos. Finally, I would like to thank Keshav for always making time for my many questions regarding wireless sensor networks.

During my stay in Waterloo, I have met many memorable individuals. Most notable is my friend Adrian who has tolerated my company with his unshakable good nature over the years. I would also like to acknowledge my colleagues in the department Qi, Jin, Aniket, Dan, Eren, Pourya, Milad, Reza, and Alejandro. Outings with Mike and Emily, Oana, Donna, and Dasha and Rodolpho added a social component that might have otherwise gone missing during my studies. Within the department, the challenges of student life were often greatly eased by the administrative efforts of Margaret Towell, Wendy Rush, Helen Jardine, and Paula Zister. Lastly, I am grateful to Mark P., Simon, Tyler and the staff at Second Cup, all of whom made this cafe such a pleasant place to work on research problems.

During my visits home, I have always enjoyed spending time with Elaine and Stan, Peter and Michelle B., Auntie Barbara and Uncle John, Annika, Jan, Auntie Lily and Uncle Bjorn, and Auntie Cheri. Likewise, correspondences with friends, both home and abroad, have been invaluable and I thank Roth and Norma, Mark S., George, Dan, Glyn, Jock, Mark E., Albert, Aaron, Jon and Lindy, Austin, and Ruth.

Above all, I wish to acknowledge my parents, Patsy and Raymond, and my brother, Roland, for their constant encouragement and support. Although we can no longer go for dim sum, my Auntie Gwen would be happy to see this finished and I am grateful for her love and support over the years. Finally, I thank Michelle Qian Z. whose kind heart always lifts my spirits and keeps me steady.

Dedication

To my parents, Patsy and Raymond Young, for their love and patience.

Contents

List of Tables	xii
List of Figures	xiv
1 Introduction	1
1.1 Motivating Attack Resistance	2
1.2 Single-Hop Communication and Resource Efficiency	4
1.2.1 A Single Sender to a Single Receiver	4
1.2.2 Multiple Senders and a Single Receiver	5
1.2.3 Multiple Senders to Multiple Receivers	6
1.2.4 Algorithmic Approaches and Commonalities	7
1.3 Summary of Main Contributions	8
1.3.1 DoS-Resistant Communication	8
1.3.2 Energy-Efficient Communication	9
1.3.3 Message-Efficient Communication	10
1.4 Thesis Roadmap	11
2 Technical Preliminaries	12
2.1 An Overview of Reliable Broadcast	12
2.1.1 Optimal Fault Tolerance in the Grid Model	13
2.1.2 Reliable Broadcast in General Topologies	16

2.2	Distributed Hash Tables and Robustness	18
2.2.1	A Review of Robust DHTs	19
3	Talk is Cheap(er): Towards Mitigating Denial-of-Service Attacks	22
3.1	Introduction	22
3.1.1	The 3-Player Scenario	23
3.1.2	Solving the 3-Player Scenario: Fair and Favourable Protocols	25
3.1.3	Our Main Contributions	25
3.1.4	Related Work	27
3.2	The 3-Player Scenario Protocol	29
3.2.1	Analysis of the 3-Party Scenario Protocol	31
3.2.2	Tolerating a Reactive Adversary	34
3.2.3	On Latency and Lower Bounds	37
3.3	Application 1: Jamming Resistance in Wireless Sensor Networks	38
3.3.1	Rationale for the 3-Player Scenario Involving WSN Devices	38
3.3.2	Local Broadcast and Guaranteed Latency	39
3.3.3	Why a Shared Schedule is Problematic	44
3.3.4	Jamming-Resistant Reliable Broadcast: Mitigating the Listening Cost Disadvantage	44
3.3.5	Reliable Broadcast in General Topologies	50
3.4	Application 2: Application-Level DDoS Attacks	57
3.4.1	Our Protocol	58
4	Reducing Listening Costs for Reliable Broadcast in Wireless Sensor Networks	61
4.1	The Bad Santa Problem	62
4.1.1	The Grid Model of Sensor Networks	63

4.2	The Bad Santa Problem and Reliable Broadcast	64
4.2.1	Utility of Las Vegas Algorithms	65
4.2.2	Byzantine Faults: Known Start Time and Source	65
4.2.3	Byzantine Faults: Unknown Start Time and Source(s)	66
4.3	Our Contributions	66
4.4	Related Work	70
4.4.1	Single-Stream Variant of the Bad Santa Problem	71
4.4.2	Multi-Stream Variant of the Bad Santa Problem	73
4.4.3	Lower Bound for Multiple Streams	74
4.4.4	An Extension to Adaptive Adversaries	75
4.4.5	Energy-Efficient Reliable Broadcast with Optimal Tol- erance	78
4.4.6	Energy-Efficient Reliable Broadcast with Near-Optimal Fault Tolerance	92
4.4.7	Unknown Start Time and Source(s)	98
4.4.8	Practical Considerations	101
5	Message-Efficient Robust Communication in Distributed Hash Tables	106
5.1	Quorums and Robustness	107
5.2	Related Work	108
5.3	Our Contributions	111
5.4	The Network Model	112
5.4.1	The Quorum Topology	112
5.4.2	Threshold Cryptography and Distributed Key Gener- ation	114
5.4.3	Spamming Attacks	116
5.4.4	Feasibility via Quorums, but Efficiency via Cryptography	117

5.5	Robust Communication Protocols	119
5.5.1	Robust Communication Protocol I	120
5.6	Determinism is an Improvement	124
5.6.1	Robust Communication Protocol II	125
5.6.2	The Join Protocols and Membership Updates	132
5.7	Microbenchmarks and Performance	134
5.7.1	Implementation and Microbenchmarks	134
5.7.2	Analysis and Discussion	137
6	Final Remarks	140
	Bibliography	162
	Appendix	163

List of Tables

4.1	Summary of frequently used notation in Chapter 4.	67
5.1	Median values of DKG completion time and CPU time per node for various s values.	136
5.2	The expected number of seconds before a quorum experiences a membership change (r_Q).	137
5.3	Median session times (in hours) derived from values for s , n_Q and r_{DKG} (in hours).	139

List of Figures

2.1	The reliable broadcast protocol of Bhandari and Vaidya [22] for tolerating Byzantine faults.	14
2.2	An illustration of the sets A_p , B_p , and B'_p	15
2.3	Pseudocode for the Certified Propagation Algorithm (CPA). . .	17
2.4	Illustration of the Chord DHT.	19
2.5	An illustration of how a Lookup operation involves forwarding information from quorum to quorum.	20
3.1	Pseudocode for 3-PLAYER SCENARIO PROTOCOL.	29
3.2	Pseudocode for LOCAL BROADCAST.	40
3.3	Pseudocode for DOS-RESISTANT RELIABLE BROADCAST. . .	46
3.4	Depiction of the sets A_p , B_p , B'_p and sister nodes for a particular node p in the grid.	50
3.5	An example of some steps of the reliable broadcast protocol for $r = 3$	51
3.6	Pseudocode for the Certified Propagation Algorithm (CPA). . .	52
3.7	Pseudocode for CPA ₀	54
3.8	Pseudocode for FCPA.	55
3.9	Pseudocode for the application of Case 2 of our 3-Player Scenario to the client-server scenario.	58
4.1	Propagation of the message by a spiral covering of the grid using corridors.	82

4.2	An illustration of the components of a corridor in the grid. . .	86
4.3	A depiction of the $(1, O(\sqrt{n}))$ Reliable Broadcast for the Byzantine Fault Model protocol for $r = 3$	89
4.4	A depiction of the $(k + 1, O(\log^{(k)}(n) + k))$ Reliable Broadcast for the Byzantine Fault Model protocol for $r = 3$ and $k = 3$. . .	95
4.5	The sets A_{p_i} , B'_{p_i} and B_{p_i} for $i = 1, 2, 3$ and $r = 3$	97
4.6	An illustration of the reliable broadcast protocol of Section 4.4.7.	102
5.1	Depiction of a DHT with quorums.	114
5.2	An overview of our general robust communication scheme. . .	120
5.3	Pseudocode for RCP-I	121
5.4	Pseudocode for RCP-II	128
5.5	An illustration of the cuckoo rule.	133

Chapter 1

Introduction

Fault tolerance is a critical issue in large-scale ad-hoc networks. Typically, such networks are characterized by a symmetric relationship between participating devices and the lack of any central authority. Network functionality is achieved in a decentralized fashion, with autonomous devices cooperating to execute tasks such as data collection, storage, and retrieval. Here, we focus on networks where the amount of resources available to each device is assumed to be both roughly equivalent and limited with respect to the system size. A large number of devices may be present; for example, large-scale systems are in existence today such as the Azureus BitTorrent network [49] and the KAD network [146] both of which see more than one million users on a daily basis. Due to the sheer size of such systems, even basic operations are complicated by the potential for faulty behaviour amongst devices.

Wireless sensor networks (WSNs) and peer-to-peer (P2P) networks are two examples of systems that capture the aspects described above. In WSNs, the devices themselves are often referred to as *motes* or *nodes*. Each node is equipped with a wireless radio and communication is often performed by broadcasting or, in the case where messages may pass through several nodes, multicasting. These networks are employed for a variety of purposes such as flood warning systems, habitat monitoring and the study of animal migratory patterns [3, 163]. Such applications require that nodes transmit and retrieve data robustly. Due to the limited radius of broadcast, routing needs to be performed using multiple hops between intermediate nodes; this is one example of a critical decentralized task that must be performed in a fault-tolerant fashion.

P2P networks share common traits with WSNs. Peers (or *nodes*) act autonomously with limited resources to provide network functionality. However, in contrast to multicasting, communication between peers often occurs as a point-to-point transmission between locations in the overlay.¹ The past decade has witnessed the advent of large-scale real-world P2P applications such as Gnutella, Kazaa, BitTorrent, and many others. While a significant amount of P2P traffic is known to involve illegal file sharing of copyrighted material [115], the P2P paradigm provides a mechanism for achieving less prosaic goals such as internet telephony [16] and data privacy [58]. Regardless of the application, data storage and retrieval are foundational operations for P2P services and it is important to that such functionality be preserved in the presence of faults.

The overarching theme of this thesis is the development of algorithms that guarantee two properties: (1) low communication overhead and (2) basic network functionality even in the face of extremely challenging attacks. Unsurprisingly, there is often a trade-off between these two properties and achieving a satisfactory balance is the focus of this thesis.

1.1 Motivating Attack Resistance

The *fail-stop model* addresses situations where component failures can occur. These failures can also model malicious attacks where system components are selectively disabled. However, a critical feature of fail-stop faults is that the affected peers are assumed to have crashed; consequently, faulty peers exhibit unresponsive behaviour which can be detected by correct peers.

In contrast, this thesis is concerned with a more general class of faults where peers that suffer a *Byzantine fault* do not necessarily crash but are controlled by an adversary. Such faulty peers are termed Byzantine and they can be used in concert to launch attacks engineered by an adversary who wishes to disrupt network functionality. This class of faults, while subsuming the fail-stop model, also captures malicious behaviour. Clearly, Byzantine faults can have a significant impact on network performance since peers do not simply disappear, but rather may persist in the system and derail crucial

¹A notable exception is P2P streaming where several peers may download content from a host simultaneously.

operations. The problem of handling such faults is non-trivial. Many algorithms, such as those for achieving leader election or broadcast (see [62] for Byzantine fault-tolerant solutions to these problems), that function in the presence of more benign failures cease to provide *any* guarantees under this more challenging fault model.

The research community is well aware of attacks on P2P systems [142, 153]. In terms of fail-stop faults, there is a vast literature on distributed hash tables (DHTs) that can tolerate random peer deletions [71, 81, 127, 134, 148, 172]. While these results may find application in certain scenarios, they fail to account for Byzantine attacks which may manifest as spamming, denial-of-service attacks (DoS), free-riding and tampering with reputation. Additionally, Byzantine faults model complex, but unintentional, failures that can occur due to the complexity of the system such as environmental factors or software errors that are not adequately modeled by the fail-stop model. In the context of more challenging attack scenarios, a P2P system is said to be *robust* if it guarantees functionality in the presence of Byzantine faults. In this area, there are many results describing robust DHTs [12–14, 31, 52, 53, 112, 135, 137, 169]. Despite the different constructions, a critical aspect of robust DHTs is secure routing between peers in the presence of Byzantine faults. However, since peers have limited resources, both in terms of bandwidth and computation, the security of the system must be balanced against the imperative of scalable communication.

In the area of WSNs, there exist several survey papers that address security issues [6, 29, 34, 84, 108, 154, 159]. In addition to traditional network security challenges, the shared communication medium of WSNs renders them vulnerable to a variety of malicious attacks [154]. A Byzantine adversary may engage in any number of activities aimed at disrupting communication such as rerouting (blackholing), message injection, wormholes, replay attacks, and others (see [34, 84] and references therein). This challenging state of affairs is further compounded by the strict energy constraints placed on the network devices themselves which are typically battery powered. Consequently, many of the standard cryptographic techniques for thwarting such attacks in the wired domain cannot be employed in WSNs. Moreover, cryptography is ineffective against an attacker that simply jams the communication medium in order to disrupt all communications within range. Finally, for many potential applications, WSNs will be deployed in hard-to-reach areas or along dangerous terrain (see [3, 163] and references therein), and once a

device has exhausted its energy supply, it is permanently inactive. Therefore, maximizing the lifetime of a device is a critical goal. Unfortunately, this goal is often at odds with the redundancy needed to overcome communication interference caused by an attacker.

1.2 Single-Hop Communication and Resource Efficiency

As discussed above, resource constraints are a critical feature of adversarial fault tolerance. The problem of achieving resource-efficient communication is a unifying theme throughout this thesis and we deal primarily with three abstract *single-hop* communication settings where a set of senders \mathcal{S} wishes to directly communicate with a set of receivers \mathcal{R} in a resource-efficient manner. We summarize these three scenarios below.

1.2.1 A Single Sender to a Single Receiver

The first single-hop communication scenario addressed is one where \mathcal{S} and \mathcal{R} each consist of a single player. The two players share a communication channel; however, there exists an adversary who aims to prevent the transmission of m by periodically blocking this channel. There are costs to send, receive or block m on the channel, and we ask: How much do the two players need to spend relative to the adversary in order to guarantee transmission of the message?

This problem abstracts many types of conflict in information networks, and the associated costs represent an expenditure of energy and network resources. For example, a particularly effective attack in WSNs is one where an adversary uses its nodes to disrupt the shared communication medium. This can occur when the attacker transmits concurrently with another (possibly legitimate) transmission such that communication is disrupted within the area of interference; this is a message collision. An adversary that launches such attacks, commonly referred to as a jamming adversary, embodies one or more Byzantine devices that attempt to interfere with communications by intentionally causing message collisions or simply flooding the channel with

useless information. Attempts by correct nodes to communicate in the presence of a jamming adversary may quickly expend their energy supply. On the other hand, a node participating in a jamming attack is also subject to energy constraints.

In this work, we approach the problem of jamming attacks from the perspective of relative energy expenditure between a correct node and the adversary. In particular, we aim to design algorithms that are efficient in the sense that (1) the expected amount of energy required by a correct node to overcome a jamming attack is less than (2) the energy expended by the adversary in pursuing a jamming strategy.

Another application of this problem setting is that of distributed denial-of-service (DDoS) attacks in the client-server model. Typically, a number of compromised clients, known collectively as a *botnet*, are employed to overwhelm a server with requests. These botnets have become commercialized with operators (also referred to as *botmasters*) renting out time to individuals for the purposes of launching attacks [54, 96]. In this setting, we seek a communication protocol for defending a server by requiring a botmaster to incur higher monetary costs in order to achieve the same level of denial-of-service it would otherwise achieve against an undefended server.

1.2.2 Multiple Senders and a Single Receiver

The second single-hop communication scenario we examine is one where \mathcal{S} consists of multiple senders who possess information m required by a single receiver in \mathcal{R} . However, many of the senders (strictly less than half) can be faulty, failing to send m or sending incorrect messages. While the majority of the senders are correct, rather than listening to all of \mathcal{S} and majority filtering on the received data, we desire an algorithm that allows the single receiver to decide on m in a more efficient manner. This single-hop scenario finds application in more general multi-hop cases where a message is propagating outward from a source node via multiple single hops. At any given time, a node which has not learned the message may sample its neighbours in order to receive it. However, these neighbors may suffer fail-stop faults or, worse, be controlled by an adversary who seeks to prevent the spread of the message.

This single-hop communication scenario finds application in the reliable broadcast problem in WSNs where a single node wishes to broadcast a mes-

sage to all other nodes in a multi-hop network. All previous results on reliable broadcast require nodes to spend significant time listening for transmissions. Surprisingly, the energy cost for listening is roughly the same as transmitting; both actions dominate the operating costs of a WSN device. Therefore, excessive listening is a critical problem in WSNs because devices are typically battery powered and, once the onboard power supply is depleted, it may be infeasible to replace it. To address this problem, we investigate how reliable broadcast can be accomplished in a more energy-efficient manner by having nodes reduce the amount of time spent in the listening state.

1.2.3 Multiple Senders to Multiple Receivers

The third single-hop communication scenario addressed in this dissertation involves \mathcal{S} and \mathcal{R} both consisting of multiple nodes. Here, a node in \mathcal{S} must cooperate with the other members of \mathcal{S} to transmit information to nodes in \mathcal{R} ; however, both sets may contain many nodes (again, strictly less than half) that are faulty. Cost is measured in the number of messages sent and received in order to obtain the necessary data from \mathcal{S} .

This generic situation arises in attack-resistant structured P2P networks where routing is typically accomplished by forwarding data along a search path consisting of peers. At each “hop” along this path, a small amount of locally available routing information is used to forward the data closer to the intended destination. The use of a small (typically logarithmic in the size of the network) amount of routing information at each peer is the core concept behind DHTs. However, routing is compromised if any peer along the search path is Byzantine. For instance, the search information may be corrupted or rerouted to an incorrect location.

A popular approach for dealing with such disruption in DHTs is to have groups of peers act in concert to overwhelm Byzantine actions. For example, if we consider a single hop, then each member may transmit the message m to the destination peer p . If the group possesses a majority of correct peers, then p can simply take the message in the majority to be correct. This idea generalizes to multiple hops and provides secure routing through the redundancy provided by using groups rather than individual peers. We use the terminology *robust communication* to describe the problem of communicating between such groups of peers. Of course, the redundancy of

this approach comes at a price. Specifically, the communication overhead is greatly increased as many more messages need to be transmitted along the search path. In this thesis, we address the problem of achieving robust communication while reducing the message complexity.

1.2.4 Algorithmic Approaches and Commonalities

While the problems addressed in this thesis span both the wired and wireless network domains, the algorithmic approaches for achieving resource efficient communication share several features in common and we discuss these below. Throughout the remainder of this thesis, we will often refer to peers or sensor network devices more generically as nodes; the network domain will always be clear from the context.

- *Randomness to Foil Adversary:* Despite behaving in a faulty manner, it is often impossible to identify Byzantine nodes since such nodes may behave correctly until there is an opportunity to derail a critical operation. Furthermore, even after the faulty behaviour, identifying the source of the problem can degenerate into an unresolvable situation of fingerprinting. Therefore, algorithms for tolerating such adversarial disruption must be *oblivious*; that is, they do not rely on having information about whether any given node is Byzantine or correct. This is a challenging situation and a natural algorithmic technique is to use randomness to foil worst-case behaviour. While we defer the details until later chapters, the results provided in this thesis make heavy use of randomized algorithms to achieve guarantees while remaining oblivious.
- *Use of Majority Action:* The number of Byzantine nodes in the system is assumed to be substantial; typically the fraction of faulty nodes in the system is a constant strictly less than $1/2$. Since the correct nodes are in the majority, it is possible to accomplish certain tasks. In a system with n nodes, a naive protocol might utilize $\Omega(n)$ communication overhead to effect an operation. For example, in a P2P system, rather than sending along a single (and more vulnerable) search path, a correct node p might send a message m to node q by asking *all* other nodes to transmit m to q . Node q can then filter on all incoming communications and take

the message in the majority to be correct. Of course, such an approach does not scale to large systems and achieving scalability requires a more sophisticated approach. Nonetheless, this example illustrates a recurring technique employed throughout this thesis that allows correct nodes to overcome adversarial disruption.

- *Efficiency Through Cryptography:* Despite the undeniable utility of cryptographic techniques, they cannot provide a satisfactory solution in many scenarios involving Byzantine faults. For example, cryptography cannot prevent wireless jamming attacks. Attempting to root out Byzantine behaviour through the use of public-key cryptography may be unwieldy in the highly dynamic and distributed settings of P2P networks. Nonetheless, there are ways in which cryptography can provide both asymptotic and practical improvements. In these cases, the adversary is assumed to be computationally bounded in the sense that certain computations (such as factoring) are assumed to be infeasible within a reasonable amount of time. This is a realistic assumption in many cases and, under this setting, we can develop algorithms that obtain substantial efficiency improvements in terms of communication complexity.

1.3 Summary of Main Contributions

In this thesis, we demonstrate algorithms that are provably correct and provide efficient communication between participants in the face of challenging adversarial attacks. In the context of the single-hop communication scenarios discussed in Section 1.2, we summarize the main contributions of this thesis below.

1.3.1 DoS-Resistant Communication

We examine our first single-hop time-slotted scenario where one player attempts to send a message m to another player over a communication channel. Assume an adversary that blocks communication for B time slots where B is *unknown* to either player and where cost is measured in terms of the number

of slots spent accessing the channel. We give a protocol guaranteeing delivery of m while the ratio of either player's expected cost to the adversary's cost is $O(B^{\varphi-2}) \approx O(B^{-0.382})$ where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio [102]. Our result implies that to prevent communication of m , the adversary incurs an asymptotically higher cost than the expected cost incurred by either correct player. Surprisingly, our result holds even in the face of an *adaptive* adversary who launches attacks using total knowledge of the past actions of all players. Moreover, in networks with a sufficient amount of communication traffic, we can tolerate a *reactive* adversary who may detect a transmission in the current time slot and then decide to jam.

As mentioned in Section 1.2, WSNs are extremely vulnerable to jamming attacks due to their shared communication medium and the constrained energy supply of the devices; such attacks constitute a type of denial-of-service attack (DoS) [108, 121]. Our protocol applies directly to a single-hop WSN where costs are measured in terms of energy expenditure for listening and sending on the wireless medium. We extend our results to obtain reliable broadcast protocols that, in comparison to previous work, offer improved resilience to jamming attacks. Finally, we apply a variant of our protocol to the case of DDoS attacks in wired networks. In the client-server model, given that the adversary uses an aggregate bandwidth of R to attack a server, the correct clients need only possess an expected aggregate bandwidth of $O(R^{0.5})$ in order to avoid zero throughput. This is useful for applications where a critical update or warning must be disseminated, and delivery to even a handful of clients is sufficient since they may then share it with others (via multicast, peer-to-peer distribution, etc.).

1.3.2 Energy-Efficient Communication

In the context of our second single-hop scenario, we address the problem of achieving reliable broadcast while minimizing energy consumption. To this end, we introduce a new data streaming problem which we call the *Bad Santa problem* described in detail in Section 4.1. Our results on this problem apply to any situation where: 1) a node can listen to a set of n nodes, out of which at least half are correct and know the correct message; and 2) each of these n nodes sends according to some predetermined schedule. We show that in order to receive the correct message with probability 1, it is necessary and sufficient for the listening node to listen to a $\Theta(\sqrt{n})$ expected number of

time slots. Moreover, if we allow for repetitions of transmissions so that each sending node sends the message $O(\log^* n)$ times, then listening to $O(\log^* n)$ expected number of time slots suffices.

We then apply our result to a popular wireless sensor network grid model. Each node is located on a point in a two dimensional grid, and whenever a node sends a message m , all awake nodes within L_∞ distance r receive m . It is known that reliable broadcast is possible if and only if $t < \frac{r}{2}(2r + 1)$ nodes within any $2r + 1$ by $2r + 1$ square can suffer Byzantine faults. These nodes are chosen and controlled by an adversary that knows everything except for the random bits of each correct node. Letting $n = r(2r + 1)$, we show how to achieve reliable broadcast that tolerates the optimal number of faults and requires each node to send and receive an expected $O(n \log^2 |m| + \sqrt{n}|m|)$ bits where $|m|$ is the number of bits in m , and, after broadcasting a fingerprint of m , each node is awake only an expected $O(\sqrt{n})$ time slots. Moreover, for $t \leq (1 - \epsilon)(r/2)(2r + 1)$, for any constant $\epsilon > 0$, we can achieve even better energy savings. In particular, if we allow each node to send $O(\log^* n)$ times, we achieve reliable broadcast with each node sending $O(n \log^2 |m| + (\log^* n)|m|)$ bits and receiving an expected $O(n \log^2 |m| + (\log^* n)|m|)$ bits and, after broadcasting a fingerprint of m , each node is awake for only an expected $O(\log^* n)$ time slots. Our results compare favorably with previous protocols that required each node to send $\Theta(|m|)$ bits, receive $\Theta(n|m|)$ bits and be awake for $\Theta(n)$ time slots.

1.3.3 Message-Efficient Communication

As previously mentioned, there are several analytical results on distributed hash tables (DHTs) that can tolerate Byzantine faults. Unfortunately, in such systems, operations such as data retrieval and message sending incur significant communication costs. For example, a simple scheme used in many Byzantine fault-tolerant DHT constructions of n nodes requires $O(\log^3 n)$ messages; this is likely impractical for real-world applications. The previous best known message complexity is $O(\log^2 n)$ *in expectation*; however, the corresponding protocol suffers from prohibitive costs owing to hidden constants in the asymptotic notation and setup costs.

In the context of our third single-hop scenario, we focus on reducing these communication costs against a computationally bounded adversary. We employ threshold cryptography and distributed key generation to define two

protocols both of which are more efficient than previous solutions. In comparison, our first protocol is *deterministic* with $O(\log^2 n)$ message complexity and our second protocol is randomized with expected $O(\log n)$ message complexity. Further, both the hidden constants and setup costs for our protocols are improved and no trusted third party is required. Finally, we present an analysis based on results from microbenchmarks conducted over PlanetLab. This analysis shows that our protocols are practical for deployment under significant levels of churn and adversarial behaviour.

1.4 Thesis Roadmap

The remainder of this document is organized as follows. Chapter 2 provides a comprehensive overview of related work, technical preliminaries, and previous results on the problems we address in this thesis. Chapter 3 describes our theoretical contributions to the problem of achieving DoS-resistant communication and illustrates how our results can be applied to concrete networking problems. Chapter 4 provides our algorithms for energy-efficient reliable broadcast along with a discussion of related practical considerations. Chapter 5 presents our theoretical results and empirical analysis regarding the problem of message-efficient robust communication in DHTs. Chapter 6 provides concluding remarks and outlines potential future research.

Finally, it is important to acknowledge that the work presented in this thesis was done in collaboration with other researchers. The content presented in Chapter 3, Chapter 4, and Chapter 5 appeared previously in [89], [87, 88], and [169], respectively. The results that are included in this document are ones with which this author was heavily involved. However, for the sake of completeness, those results which do not meet this criterion have been included in an appendix at the end of this thesis. Specifically, included in this appendix are two lemmas dealing with lower bounds for the Bad Santa problem dealt with in Chapter 4.

Chapter 2

Technical Preliminaries

As discussed in Chapter 1, the results for the single-hop communication problems can be extended to provide solutions to communication problems involving multiple hops. One such problem that figures prominently in both Chapter 3 and Chapter 4 is that of *reliable broadcast*. In this chapter, we provide a comprehensive overview of the previous results on this problem in order to provide the necessary background for understanding our work. In Chapter 5, we address the problem of tolerating adversarial attacks in structured peer-to-peer networks; specifically, in distributed hash tables (DHTs). This is an area that has received significant attention in the literature and we review several previous works in order to provide the context for our new results.

2.1 An Overview of Reliable Broadcast

A fundamental problem in distributed systems is Byzantine Agreement. Since its introduction by Pease *et al.* [117], this problem has received much attention. In Byzantine Agreement, there are a total of n devices in the system. Of these n devices, t of them may deviate from protocol in an arbitrary fashion; such devices are termed Byzantine or faulty. Furthermore, these devices are assumed to be controlled by an adversary and, therefore, they may act in concert to cause trouble for the network. The remaining $n - t$ devices that are not controlled by the adversary are assumed to obey protocol and we call these devices *correct*. Note that while the devices corrupted by the

adversary are fixed prior to the execution of any protocol (a static adversary), the correct devices do not know which are faulty. A device p_i can set a value field to either 0 or 1 and must commit to a value in $\{0, 1\}$ by the end of the protocol. There exists a dealer holding an initial value $v \in \{0, 1\}$. A protocol that achieves Byzantine Agreement is one which guarantees the following two properties. First, all correct devices commit to the same value in $\{0, 1\}$. Second, if the dealer is correct, then all correct devices commit to v . Therefore, this is a question of secure broadcast. While these are basic guarantees one would desire in such an adversarial setting, the problem of achieving them is non-trivial due to the behaviour of the Byzantine devices.

In this thesis, we address the problem of Byzantine Agreement in WSNs. In this domain, the problem has typically been labeled as *reliable broadcast* in the literature and this is the terminology we use throughout. In this context, we refer to *nodes* rather than *devices*. Here, our focus is on algorithms for guaranteeing reliable broadcast while addressing the challenging energy constraints placed on WSN devices.

2.1.1 Optimal Fault Tolerance in the Grid Model

We begin by describing the popular grid model of WSNs [19, 21–23, 87, 88, 90, 150]. In this model, each point in a two-dimensional grid corresponds to a node. Let $p(x, y)$ denote the node p at location (x, y) in the grid. Whenever a node sends a message, all listening nodes within L_∞ distance r receive the message.¹ We will use $N(p)$ or $N(x, y)$ to denote the set of nodes within radius r of a node $p(x, y)$; this is p 's *broadcast neighbourhood*. Within any broadcast neighbourhood, at most t nodes may suffer a fault.

For reliable broadcast in the grid, the dealer (who is also referred to as the *source* in the literature) is located at coordinates $(0, 0)$ and all nodes know this. There is a global broadcast schedule that assigns to each node p a time slot in which p may broadcast. An example of a broadcast schedule is given in [90]: in each round, each node in position (x, y) broadcasts in time slot $((x \bmod (2r + 1)) \times (2r + 1) + (y \bmod (2r + 1))) \bmod (2r + 1)^2$. For simplicity, messages are assumed to fit within a single slot

¹The distance between two points (x_1, y_1) and (x_2, y_2) in the L_∞ metric is $\max\{|x_1 - x_2|, |y_1 - y_2|\}$.

RELIABLE BROADCAST PROTOCOL (BHANDARI AND VAIDYA, 2005)

- Initially, the source s does a local broadcast of message m .
- Each node $i \in N(s)$ commits to the first value it receives from s and does a one-time broadcast of $\text{COMMIT}(i, m)$.
- The following protocol is executed by each node j (including those nodes in the previous two steps):
 - On receipt of a $\text{COMMIT}(i, m)$ message from a neighbor i , node j records the message and broadcasts $\text{HEARD}(j, i, m)$.
 - On receipt of a $\text{HEARD}(j', i, m)$, node j records this message.
 - Upon receiving COMMIT or HEARD messages that 1) claim v as the correct value and 2) are received along at least $t + 1$ node disjoint paths that all lie within a single neighbourhood, then node j commits to v and does a one-time broadcast of $\text{COMMIT}(j, m)$.

Figure 2.1: The reliable broadcast protocol of Bhandari and Vaidya [22] for tolerating Byzantine faults.

The work of Koo [90], and the subsequent work by Bhandari and Vaidya [22], provides tight lower and upper bounds on the number of faults t . In particular, when faults are Byzantine, reliable broadcast is possible in the grid if and only if $t < (r/2)(2r + 1)$. Figure 2.1 provides the pseudocode for the reliable broadcast protocol of Bhandari and Vaidya [22] that achieves the maximum (or optimal) fault tolerance.

Proving that this protocol is correct is non-trivial. Here, we summarize the proof; however, the reader is referred to [22] for a complete analysis. In the protocol itself, the message $\text{COMMIT}(i, m)$ signifies that node i has committed to a message m , and the message $\text{HEARD}(j, i, m)$ signifies that node j has heard a message $\text{COMMIT}(i, m)$. We require the notion of a perturbed neighbourhood $PN(p)$ of $p(a, b)$ as $PN(p) = N(a + 1, b) \cup N(a - 1, b) \cup N(a, b + 1) \cup N(a, b - 1)$. The proof in [22] works by showing that for each node p in $PN(a, b) - N(a, b)$, there exist $2t + 1$ paths P_1, \dots, P_{2t+1} belonging to a single neighbourhood $N(a, b + r + 1)$, each having one of the

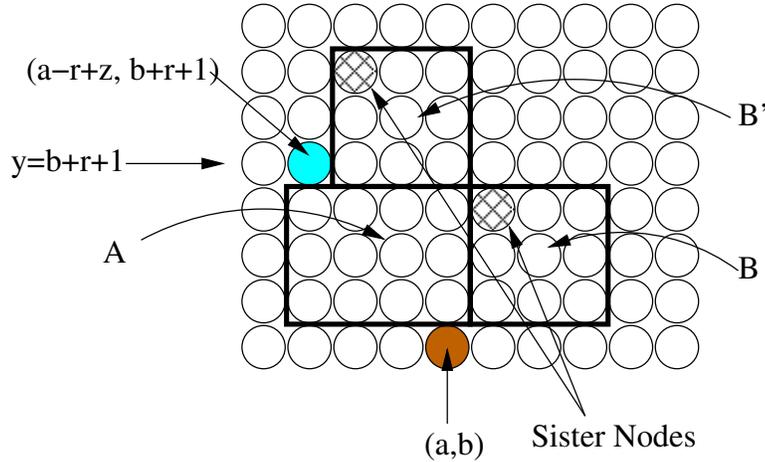


Figure 2.2: An illustration of the sets A_p , B_p , and B'_p where $z = 0$, $r = 3$ and $a, b = 0$. Node p is located at position $(a - r + z, b + r + 1)$. A pair of sister nodes, one in B' and the other in B , are highlighted.

forms listed below:

- $P_i = (q, p)$ which is a one hop path $q \rightarrow p$ or
- $P_i = (q, q', p)$ which is a two hop path $q \rightarrow q' \rightarrow p$

where q, q', p are distinct nodes and q, q' lie in a single neighbourhood $N(a, b + r + 1)$, and $q \in N(a, b)$ where, critically, nodes in $N(a, b)$ have committed to the correct message. The existence of these $2t + 1$ paths, and the fact that each broadcast neighbourhood has at most $t < (r/2)(2r + 1)$ Byzantine faults, is sufficient to prove that reliable broadcast is achieved by the protocol. For simplicity, we can consider $p \in N(a, b + 1)$ since the analysis is nearly identical for the cases where $p \in N(a + 1, b)$, $p \in N(a - 1, b)$, and $p \in N(a, b - 1)$.

The node p lies in $N(a, b + 1) - N(a, b)$ and can be considered to have location $(a - r + z, b + r + 1)$ where $0 \leq z \leq 2r$. Now, summarizing the proof in [22], we demonstrate that there exist $r(2r + 1)$ node-disjoint paths $P_1, \dots, P_{r(2r+1)}$ all lying within the same neighbourhood:

- **One-Hop Paths:** the set of nodes $A_p = \{q(x, y) \mid (a - r) \leq x \leq (a + z) \text{ and } (b + 1) \leq y \leq (b + r)\}$ lie in $N(a, b)$ and are neighbors of p . Therefore, there are $r(r + z + 1)$ paths of the form $q \rightarrow p$ where $q \in A_p$.
- **Two-Hop Paths:** consider the sets $B_p = \{q(x, y) \mid (a + z + 1) \leq x \leq (a + r) \text{ and } (b + 1) \leq y \leq (b + r)\}$ and $B'_p = \{q'(x', y') \mid (a + z + 1 - r) \leq x' \leq a \text{ and } (b + r + 1) \leq y' \leq (b + 2r)\}$. The nodes in B_p lie in $N(a, b)$ while the nodes in B'_p lie in $N(p)$. Moreover, the set B'_p is obtained by shifting left by r units and up by r units. Therefore, there is a one-to-one mapping between the nodes in B_p and the nodes in B'_p . For $u \in B_p$, we will call the corresponding node $u' \in B'_p$, the *sister node* of u . Note that each node has at most two sister nodes; this can be seen in Figure 4.2. Hence, there are $r(r - z)$ paths of the form $q \rightarrow q' \rightarrow p$.

Therefore, there are a total of $r(r + z + 1) + r(r - z) = r(2r + 1)$ disjoint paths all lying in a single neighbourhood $N(a, b + r + 1)$. Figure 2.2 illustrates aspects of the discussion above where $a, b = 0$. Now, note that the predecessor set $G_p = A_p \cup B'_p$ is the set of nodes to which p must listen in order to gather information that will allow it to commit to the correct message.

In Chapter 3 and Chapter 4, we extend our results for single-hop communication problems to achieve protocols for reliable broadcast. Our protocols rely heavily on the results of Bhandari and Vaidya [22] discussed above. In particular, we assume that each node p knows a predecessor set G_p of nodes to which node p should listen for messages. As we have just reviewed, the existence of G_p is shown by the constructive proofs in [22]. Our protocols specify when each node p should listen to nodes in G_p and when each node p should broadcast the message to which it has committed; the details of how this is accomplished are discussed later.

2.1.2 Reliable Broadcast in General Topologies

Pelc and Peleg [119] examine a generalization of the t -locally bounded fault model; that is, where each node contains at most t Byzantine nodes within its neighbourhood. Specifically, they examine the broadcast protocol of Koo [90], which the authors call the *Certified Propagation Algorithm* (CPA), with the aim of establishing conditions for which it achieves reliable broadcast under arbitrary graphs in contrast to the grid model. Again, CPA addresses

Certified Propagation Algorithm (Koo [90] and Pelc and Peleg [119])

- The dealer d sends the message m to all of its neighbors and terminates.
- For a correct node $u \in N(d)$, upon receiving m from d it commits to m , node u announces this commitment of its neighbors and terminates.
- If a node is not a neighbor of the source, then upon receiving $t + 1$ copies of m from $t + 1$ distinct neighbors, it commits to m , and announces this commitment to its neighbors and terminates.

Figure 2.3: Pseudocode for the Certified Propagation Algorithm (CPA).

the case where all nodes obey a global broadcast schedule (i.e. there is no jamming adversary). The authors define $X(p, d)$ to be the number of nodes in p 's neighbourhood $N(p)$ that are closer to d than p and then introduce the parameter $X(G) = \min\{X(p, d) \mid p, d \in V, (p, s) \notin E\}$.

Using the parameter $X(G)$, Pelc and Peleg [119] prove that, for *any* graph G with dealer d such that $t < X(G)/2$, CPA achieves reliable broadcast; the pseudocode is given in Figure 2.3. The intuition behind the correctness of CPA is as follows. If CPA fails to achieve reliable broadcast, then there is some node p that fails to commit on the correct message m ; let p be the peer closest to the dealer who has committed incorrectly. However, there are at least $2t + 1$ nodes closer to d than p to which p can listen. At most t of these nodes are faulty; therefore, p can listen to a set Γ_p of at least $t + 1$ nodes that are correct. Since p has committed incorrectly, it follows that at least one correct node in Γ_p has committed incorrectly, (otherwise, p would have received $t + 1$ correct messages). However, this contradicts the fact that p is the first correct node to fail in this fashion; this contradiction completes the argument.

We note that CPA does not necessarily tolerate the optimal number of faults (see [168]). CPA is considered in Chapter 3 as an example that our single-hop result can be incorporated into more general topologies. We note that Ichimura and Shigeno [77] establish a different parameter for deriving bounds on t ; however, this is far more involved and the application of our results to general topologies is convincingly shown using the CPA algorithm.

2.2 Distributed Hash Tables and Robustness

In Chapter 5, we address the topic of adversarial fault tolerance in DHTs. A distributed hash table (DHT) is a structured peer-to-peer network which provides for scalable and distributed storage and retrieval of data items. The literature contains many different proposals for DHTs (see e.g. [127, 148, 172]). Despite the distinctions between various DHTs, the standard paradigm consists of a secure hash function $h(\cdot)$ which is used to map objects to a *keyspace*. The objects being mapped by $h(\cdot)$ are peers and data items and the keyspace is an abstract interval often scaled to be within $[0, 1)$.

To provide an example, we give an overview of the Chord DHT [41, 148]. For convenience, the “key space” of Chord is assumed to be scaled so it is in the range $[0, 1)$ and Chord can be viewed as a ring of unit circumference. Here, the value 0 lies at the twelve o’clock position on the ring and the identifier space increases moving in a clockwise direction. All peers in Chord have identifiers which are points on the ring. Chord provides one basic operation: $\text{Lookup}(\cdot)$. For a location k on ring, $\text{Lookup}(k)$ returns the peer, p , whose location on the ring minimizes the clockwise distance between k and p . Typically, k represents a key for a data item and $\text{Lookup}(k)$ is the peer responsible for storing that data item.

Chord implements the operation $\text{Lookup}(\cdot)$ in the following way. All peers in the network are assumed to know some number M which is always greater than the number of peers in the network. For a location p on the ring and integer i between 1 and $\log M - 1$, let $f(p, i) = p + 2^i/M$ (modulo 1) be a location on the ring. For each i between 1 and $\log M - 1$, each peer p maintains a link to the peer whose peer point is closest clockwise to the point $f(p, i)$; this is often referred to as a *finger link*. Peer p also keeps a link to the peer closest clockwise from p stored as $f(p, 0)$. When a peer p links to a peer p' , the peer p simply stores the necessary information such as the IP address and port of p' . For points p and k on the ring, let $\text{next}(p, k)$ be the point in the set $\{f(p, 0), f(p, 1), f(p, 2), \dots, f(p, \log M - 1)\}$, which has closest clockwise distance to k . The number of unique peers that a peer p links to is $O(\log n)$. Figure 2.4(a) illustrates $\text{next}(p, i)$ links.

The $\text{Lookup}(\cdot)$ operation works as follows. Assume that peer p calls $\text{Lookup}(k)$ for a key k . If $\text{next}(p, k) = p$, then p already knows the successor of k : it is simply the closest clockwise peer to p . The search terminates by

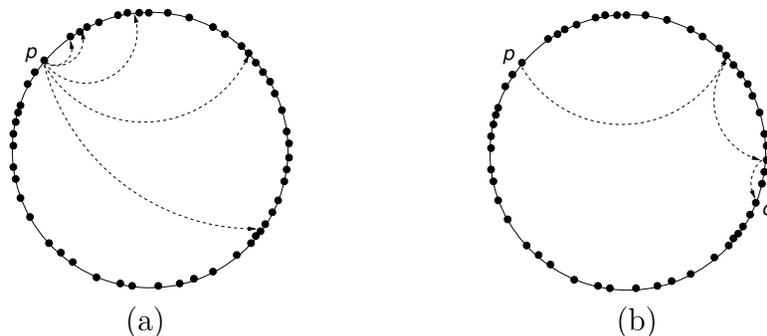


Figure 2.4: Illustration of the Chord DHT. (a) The $next(p, i)$ links held by peer p . (b) The path taken by a `Lookup` operation where k is held by the peer q .

returning this peer. If $next(p, k) = p'$ where $p' \neq p$, then p forwards the search request to p' . This greedy procedure is repeated until the search terminates in $O(\log n)$ hops. Figure 2.4(b) illustrates a query initiated by peer p and terminating at peer q which is responsible for holding key k .

2.2.1 A Review of Robust DHTs

There is a large body of literature on tolerating adversarial faults in DHTs [12–14, 52, 53, 75, 112]; such constructions are sometimes referred to as *robust* DHTs. Almost all such results make use of *quorums*, which are sets of $\Theta(\log n)$ peers with the property that a minority of the peers in a quorum have suffered Byzantine faults. Using quorums, it is possible to overcome adversarial behaviour by having the members of a quorum use all-to-all communication with the members of the next quorum for each hop in a `Lookup` operation. If the Byzantine peers attempt to deviate from protocol, this adversarial behaviour can be overwhelmed through majority action; that is, at each hop, each correct peer majority filters on incoming messages. In this way, even if the Byzantine peers corrupt or drop messages, the correct peers in each quorum will still forward the query and, therefore, guarantee the `Lookup` operation succeeds.

Figure 2.5 illustrates the general quorum structure; note that quorums overlap. However, the use of quorums can fail if the adversary can obtain a

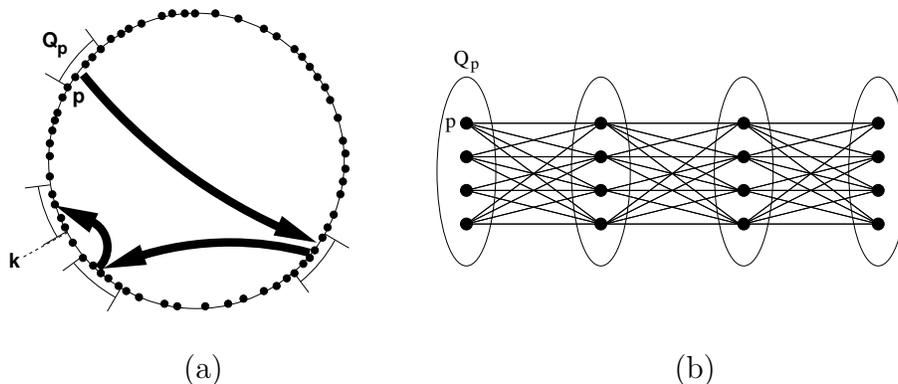


Figure 2.5: An illustration of how a Lookup operation involves forwarding information from quorum to quorum. (a) Peer p issues a request to its quorum Q_p for an item with key k . This request is forwarded using all-to-all communication between quorums until the key is found. (b) All-to-all communication between quorums. Each peer listens to all the messages it receives. Each peer then majority filters on these messages and takes the one in the majority to be correct.

majority of corrupted peers in any quorum. This can happen if an adaptive adversary has its peers join and leave the network until a favourable placement is attained. For instance, the adversary may target a quorum in the following manner. The adversary adds a corrupted peer p into the system to see where it is placed. If p lands within the target quorum, the adversary keeps the peer active in the system; otherwise, p leaves. Over a number of join and leave events the adversary may accumulate a large number of peers in the target quorum, eventually obtaining a majority.

The state of the art in protecting against such insider attacks is due to results by Awerbuch and Scheideler [12, 13] which describe a DHT that remains robust even if the number of join and leave events is polynomial in the size of the network n . In [14], the authors extend this work to the case where the attacker can even force correct nodes to leave the system temporarily.

In [112], all-to-all communication is used to pass messages between quorums. In both [13] and [14], communication between quorums is not specified in detail, although all-to-all communication suffices. In such settings, com-

munication between two quorums requires $O(\log^2 n)$ messages. In this case, a typical implementation of a **Lookup** operation requires $O(\log^3 n)$ messages. Work by Saia and Young [135] shows how such message passing can be accomplished more efficiently, asymptotically reducing the costs of employing such quorums.

Despite the significant progress on using quorums to tolerate Byzantine attacks, these results are still likely impractical for real-world deployment due to the communication overhead they incur. In Chapter 5, we address this issue by devising algorithms that further reduce the costs of communication. Specifically, our algorithms achieve asymptotic improvements in the number of messages required (the *message complexity*) for executing a **Lookup** operation and provide the first steps towards achieving practical fault tolerance through quorums.

Chapter 3

Talk is Cheap(er): Towards Mitigating Denial-of-Service Attacks

3.1 Introduction

Denial-of-service (DoS) attacks are an increasingly common threat to the availability of large-scale networks. In the wireless domain, practitioners have expressed concerns that important surveillance services will suffer DoS attacks by the deliberate disruption of the communication medium [39] and there is a large body of literature on various practical security concerns (see [108, 121] and references therein). In the wired domain, the distributed denial-of-service (DDoS) attacks in November of 2010 on Amazon, Visa, Mastercard, and PayPal [18, 103, 151] demonstrated that even prominent companies are vulnerable over the Internet. Given this state of affairs, an interesting question arises: Is it fundamentally “harder” or “easier” to communicate in such large-scale networks than it is to block communication?

To analyze this question from an algorithmic perspective, we define the following simple problem, which we call the *3-Player Scenario*: Alice wishes to guarantee transmission of a message m directly to Bob over a single communication channel. However, there exists an adversary Carol who aims to prevent communication by blocking transmissions over the channel. We consider two cases: (Case 1) when Carol may spoof or even control Bob, which

allows her to manipulate an unwitting Alice into incurring excessive sending costs; and (Case 2) where Bob is both correct and unspoofable, and his communications cannot be blocked. Here, “cost” corresponds to a network resource, such as energy in wireless sensor networks (WSNs) or bandwidth in wired networks.

In the 3-Player Scenario, we show that communication is fundamentally cheaper than preventing communication. Specifically, we describe a protocol that guarantees correct transmission of m , and given that Carol incurs a cost of B , has the following properties. In Case 1, the expected cost to both Alice and Bob is $O(B^{\varphi-1} + 1)$ where φ is the golden ratio. In Case 2, the expected cost to both Alice and Bob is $O(B^{0.5} + 1)$. In both cases, Carol’s cost asymptotically exceeds the expected cost of either correct player. Finally, we note that these two cases differ significantly in terms of how the adversary may interfere with communications; therefore, it is not possible to achieve a middle ground between these two relative costs.

3.1.1 The 3-Player Scenario

The 3-Player Scenario represents our first single-hop communication scenario as discussed in Chapter 1. We now describe the critical model parameters of the 3-Player Scenario.

Las Vegas Property: Communication of m from Alice to (a correct) Bob must be guaranteed with probability 1; that is, we require a Las Vegas protocol for solving the 3-Player Scenario. An obvious motivation for this Las Vegas property is a critical application, such as an early warning detection system or the dissemination of a crucial security update, where minimizing the probability of failure is paramount. The Las Vegas property has additional merit in multi-hop WSNs where Monte Carlo algorithms may not be able to achieve a sufficiently low probability of error; we expand on this in Section 3.3.5.

Channel Utilization: Sending or listening on the communication channel by Alice and Bob is measured in discrete units called *slots*. For example, in WSNs, a slot may correspond to an actual time slot in a time division multiple access (TDMA) type access control protocol. The cost for sending or listening is S or L per slot, respectively. When Carol blocks a slot, she disrupts the channel such that no communication is possible; blocking costs

J per slot. If a slot contains traffic or is blocked, this is detectable by a player who is *listening* at the *receiving end* of the channel, but not by the originator of the transmission. For example, a transmission (blocked or otherwise) from Bob to Alice is detectable only by Alice; likewise, a transmission (blocked or otherwise) from Alice to Bob is detectable only by Bob. A player cannot discern whether a blocked slot has disrupted a legitimate message; only the disruption is detectable. For example, high-energy noise is detectable over the wireless channel in WSNs, but a receiving device cannot tell if this results from a message collision or a device deliberately disrupting the channel. We let B be the total amount Carol will spend over the course of the algorithm; this value is *unknown* to either Alice or Bob. Finally, we say that any player is *active* in a slot if that player is sending, listening or blocking in that slot.

Correct and Faulty Players: If Alice is faulty, there is clearly no hope of communicating m ; therefore, Alice is assumed to be correct. Regarding the correctness of Bob, in Case 1, Carol may spoof or control Bob; in Case 2, communications from Bob are always trustworthy. We emphasize that, in Case 1, Alice is uncertain about whether to trust Bob since he may be faulty. This uncertainty corresponds to scenarios where a trusted dealer attempts to disseminate content to its neighbors, some of whom may be faulty and attempt to consume resources by requesting numerous retransmissions. Case 2 corresponds to situations where communications sent by Bob are never disrupted and can be trusted; here, the blocking of m being sent from Alice to Bob is the only obstacle.

Types of Adversary: Carol has full knowledge of past actions by Alice and Bob. This allows for *adaptive attacks* whereby Carol may alter her behaviour based on observations she has collected over time. Furthermore, under conditions discussed in Section 3.2.2, Carol can also be *reactive*: in any slot, she may detect a transmission and then disrupt the communication, however, we assume that she cannot detect when a player is listening. This is pertinent to WSNs where the effectiveness of a reactive adversary has been shown experimentally.

3.1.2 Solving the 3-Player Scenario: Fair and Favourable Protocols

We analyze the cost of our algorithms as a function of B . In this way, we obtain a notion of cost incurred by a player that is *relative to the cost incurred by Carol*. In devising our algorithms, we seek to achieve two properties with regards to relative cost.

First, our protocol should be *fair*; that is, Alice and Bob should incur the same *worst case asymptotic cost* relative to the adversary. When network devices have similar resource constraints, such as in WSNs where devices are typically battery powered, this is critical. Alternatively, in networks where a collection of resource-scarce devices (i.e. client machines represented by Alice) occupy one side of the communication channel and a single well-provisioned device (i.e. a server represented by Bob) occupies the other side, the *aggregate* cost to Alice's side should be roughly equal to that of Bob.

Second, we desire *favourable* protocols; that is, for B sufficiently large, Alice and Bob both incur asymptotically less expected cost than Carol. DoS attacks are effective because a correct device is *always* forced to incur a higher cost relative to an attacker. However, if the correct players incur asymptotically less cost than Carol, then Alice and Bob enjoy the advantage, and Carol is faced with the problem of having her resources consumed disproportionately in her attempt to prevent communication.

3.1.3 Our Main Contributions

Throughout, let $\varphi = (1 + \sqrt{5})/2$ denote the golden ratio. We also draw attention to the well-known relationship that $\Phi = \varphi - 1 = 1/\varphi \approx 0.62$ where Φ is known as the *golden ratio conjugate*. Our results can be presented using Φ rather than φ ; however, for readability, we utilize φ throughout. We assume that S , L , and J are fixed constants. Our main analytical contributions are listed below.

Theorem 1. *Assume Carol is an adaptive adversary and that she is active for B slots. There exists a fair and favourable algorithm for the 3-Player Scenario with the following properties:*

- In Case 1, the expected cost to each correct player is $O(B^{\varphi-1} + 1) = O(B^{0.62} + 1)$. In Case 2, the expected cost to each correct player is $O(B^{0.5} + 1)$.
- If Bob is correct, then transmission of m is guaranteed and each correct player terminates within $O(B^\varphi)$ slots in expectation.

In networks with sufficient traffic, Theorem 1 still holds when Carol is also reactive (Section 3.2.2). We also prove that any protocol which achieves $o(B^{0.5})$ expected cost for Bob requires more than $2B$ slots to terminate (Section 3.2.3); this lower bound has bearing on the worst-case $\omega(B)$ slots required by our protocol.

Our next Theorems 2 and 3 are applications of Case 1 of Theorem 1 to WSNs. We consider a more general setting where Alice wishes to locally (single-hop) broadcast to n neighboring receivers of which any number are spoofed or controlled by Carol. Unfortunately, a naive solution of having each receiver execute a separate instance of our 3-Player Scenario protocol fails to be fair. Thus, we need a different algorithm to achieve the following result.

Theorem 2. *There exists a fair (up to small polylogarithmic factors in n) and favourable algorithm for achieving local broadcast with the following properties:*

- If Carol's receivers are active for a total of B slots, then the expected cost to Alice is $O(B^{\varphi-1} \ln n + \ln^\varphi n)$ and the expected cost to any correct receiver is $O(B^{\varphi-1} + \ln n)$.
- Transmission of m is guaranteed and all correct players terminate within $O((B + \ln^{\varphi-1} n)^{\varphi+1})$ slots (not in expectation). For $B \geq \ln^{\varphi-1} n$, this is within an $O(B^\varphi)$ -factor of the optimal latency.

Since the adversary may simply jam for the first B slots, it is impossible to achieve communication in less than $B + 1$ slots; it is this value to which we refer when speaking of the *optimal latency*. Reliable broadcast in *multi-hop* WSNs deals with conveying m from one node to all other nodes in the network. We make the standard assumptions that any node p can be heard by the set of neighboring nodes in the topology, $N(p)$ and that, for any p ,

at most t nodes in $N(p)$ suffer a fault (t -bounded fault model) [21, 22, 90]. We analyze the grid model using the result of Bhandari and Vaidya [22], and general graphs using the Certified Propagation Protocol (CPA) protocol of Pelc and Peleg [119].

Theorem 3. *For each correct node p , assume the t nodes in $N(p)$ are Byzantine and can be used by Carol to disrupt p 's communications for $\beta \leq B_0$ time slots where B_0 is a known positive value. Then, using the local broadcast protocol of Theorem 2, fair and favourable reliable broadcast is possible under the following topologies:*

- *In the grid with the optimal fault tolerance $t < (r/2)(2r + 1)$.*
- *In any graph, assuming that (a) t is appropriately bounded such that CPA achieves reliable broadcast and (b) the topology and location of the dealer is known to all nodes.*

To the best of our knowledge, all previous reliable broadcast protocols require correct nodes to spend more energy in communication attempts than that spent by adversarial nodes. Our results are the first favourable protocols and, importantly, the first to account for the *significant cost of listening* to the wireless channel.

Finally, Theorem 4 is an application of Case 2 of Theorem 1 to a client-server scenario where Carol represents malicious clients engaging in a DDoS attack on a server.

Theorem 4. *Assume Carol commits her DDoS attack using a bandwidth R . Then, zero throughput is avoided if the expected aggregate bandwidth (upstream or downstream bits per second) of both the clients and the server is $G = O(R^{0.5})$, and the probability of a serviced request is $G/(G + R)$.*

Therefore, against a server defended by our protocol, Carol must incur additional monetary costs in order to procure the number of machines necessary for sustaining the level of attack she would otherwise achieve.

3.1.4 Related Work

Jamming Attacks in WSNs: Several works addressing applied security considerations demonstrate that devices in a WSN are vulnerable to

adversarial jamming [8, 17, 99, 165] where the adversary deliberately disrupts the communication medium. Defenses include spread spectrum techniques (frequency or channel hopping), mapping with rerouting, and others (see [101, 113, 162, 164] and references therein).

There are a number of theoretical results on jamming adversaries; however, none explicitly accounts for listening costs and there is no notion of favourability. Gilbert *et al.* [62] examine the duration for which communication between two players can be disrupted in a model with collision detection in a time-slotted network against an adversary who interferes with an unknown number of transmissions. As we do here, the authors assume channel traffic is always detectable at the receiving end (i.e. silence cannot be “forged”). Pelc and Peleg [120] examine an adversary that randomly corrupts messages; we do not require the adversary to behave randomly. Awerbuch *et al.* [11] give a jamming-resistant MAC protocol in a single-hop network with an adaptive, but non-reactive, adversary. Richa *et al.* [130] significantly extend this work to multi-hop networks. Dolev *et al.* [46] address a variant of the gossiping problem when multiple channels are jammed. Gilbert *et al.* [61] derive bounds on the time required for information exchange when a reactive adversary jams multiple channels. Meier *et al.* [104] examine the delay introduced by a jamming adversary for the problem of node discovery, again in a multi-channel setting. Dolev *et al.* [47] address secure communication using multiple channels with a non-reactive adversary. Recently, Dolev *et al.* [45] consider wireless synchronization in the presence of a jamming adversary. We refer the reader to the survey by Young and Boutaba [168] for a more in-depth review of this area.

Reliable Broadcast: Reliable broadcast has been extensively studied in the grid model [19, 21–23, 87, 88, 90, 150]. Listening costs are accounted for by King *et al.* [87, 88] but jamming adversaries are not considered; however, the authors introduce the *Bad Santa* problem which we use to achieve a lower bound result in Section 3.2.3. With a reactive jamming adversary, Bhandhari *et al.* [24] give a reliable broadcast protocol when the amount of jamming is bounded and known *a priori*; however, correct nodes must expend considerably more energy than the adversary. Progress towards fewer broadcasts is made by Bertier *et al.* [20]; however, each node spends significant time in the costly listening state. Alistarh *et al.* [4] assume collision detection and achieve non-cryptographic authenticated reliable broadcast. They apply their result to the grid model with a reactive jamming adversary; however,

<p>3-PLAYER SCENARIO PROTOCOL for round $i \geq 2$</p> <p><i>Send Phase:</i> For each of the 2^{ci} slots do</p> <ul style="list-style-type: none"> • Alice sends m with probability $2/2^i$. • Bob listens with probability $2/2^{(c-1)i}$. <p>If Bob received the message, then Bob terminates.</p> <p><i>Ack Phase:</i> For each of the 2^i slots do</p> <ul style="list-style-type: none"> • Bob sends a <code>req</code> message. • Alice listens with probability $4/2^i$. <p>If Alice listened to a slot in the Ack Phase where no <code>req</code> message or blocking was detected, she terminates.</p>

Figure 3.1: Pseudocode for 3-PLAYER SCENARIO PROTOCOL.

in their algorithm, nodes incur considerable listening costs.

Application-Level DDoS Attacks: At the application layer, DDoS attacks typically involve attackers masquerading as legitimate clients by sending a large volume of proper requests with the aim of overwhelming the computational resources of a server. This is in contrast to flooding attacks which achieve disruption by depleting the bandwidth of the network; such attacks typically require significant bandwidth and attackers are more easily identified due to the out-of-band traffic. Application-level DoS attacks are common with recorded attacks on high-profile companies such as Yahoo, Amazon, CNN, eBay, and many others [56]. Proposals for dealing with DDoS attacks include over-provisioning [125], throttling techniques [63, 107], and currency schemes (see [10, 80, 152] and references therein). In currency schemes, the server provides service only to a client who pays in some form of currency. In [152], bandwidth is used as currency and, if the clients' aggregate bandwidth exceeds that of the attackers, then the clients capture server resources. Our work is complementary by delineating bounds on the expected bandwidth required to guarantee that the correct clients avoid zero throughput.

3.2 The 3-Player Scenario Protocol

Figure 3.1 gives the pseudocode for our protocol called 3-PLAYER SCENARIO PROTOCOL (3PSP). Each round $i \geq 2$ consists of 2 phases and c is a constant

to be determined later. We summarize a round i :

- *Send Phase*: This phase consists of 2^{ci} slots. In each slot: Alice sends m with probability $\frac{2}{2^i}$ for an expected total of $2^{(c-1)i+1}$ slots and Bob listens with probability $\frac{2}{2^{(c-1)i}}$ for an expected total of 2^{i+1} slots.
- *Ack Phase*: This phase consists of 2^i slots. If Bob has not received m , then Bob sends a request for retransmission, **req**, for all 2^i slots. Alice listens in each slot with probability $4/2^i$ (note that $i \geq 2$ is required) for an expected total 4 slots.

Termination Conditions: Termination conditions are important because Carol cannot be allowed to keep the players active in perpetuity while simultaneously forcing them to incur a *higher* cost. Bob terminates the protocol upon receiving m . Since Alice is not spoofed, as discussed in Section 3.1.1, this termination condition suffices. Alice terminates if she listens to a slot in the Ack Phase which is not blocked and does not contain **req** message; since blocked slots are detectable by Alice (who is on the receiving end of a **req** message) while listening (Section 3.1.1), this condition suffices. That is, Alice continues into the next round if and only if (1) Alice listens to zero slots or (2) all slots listened to by Alice in the Ack Phase contain a blocked slot or **req**. We highlight the two situations where this condition is met:

- *Send Failure*: Bob is correct and has not received m .
- *Ack Failure*: Bob is faulty and sends **reqs**, *or* Bob is correct and terminated and Carol either spoofs **reqs** or blocks slots in order to trick Alice into thinking a valid **req** was indeed sent and/or blocked.

Ack Failures and Cases 1 & 2: Note that an “acknowledgement” occurs via silence in at least one slot in the Ack Phase. We say an *Ack Failure* occurs when Carol blocks for all slots in the Ack Phase.

In Case 1, an Ack Failure corresponds to a critical attack that can be employed in Ack Phase after the delivery of m . Carol can avoid the listening costs in the Send Phase, and then drain Alice’s energy by making it appear as if Bob repeatedly did not receive m and is requesting a retransmission in the Ack Phase. This attack affects Alice only. Note that if Bob is actually correct, the attack is only effective once m is received since, if a correct Bob has not received m , a **req** will be issued anyway and the attack accomplishes nothing.

In Case 2, no blocking occurs in the Ack Phase and, therefore, no Ack Failure can occur. In fact, in Case 2, the Ack Phase can be shortened to a single slot where Bob sends his `req` and Alice listens; however, this does not change our cost analysis and our current presentation is more general.

3.2.1 Analysis of the 3-Party Scenario Protocol

For a given round, we say it is a *send-blocking* round if Carol blocks at least half of the slots in the Send Phase; otherwise, it is a *non-send-blocking* round. Similarly, a *ack-blocking* round is a round where Carol blocks or spoofs `req` messages from Bob in at least half the slots in the Ack Phase; otherwise, it is *non-ack-blocking*. Throughout, assume ceilings on the number of active slots of a player if it is not an integer.

Bounds on c : Clearly, $c > 1$ or Bob's listening probability in the Send Phase is nonsensical. For Case 1, note that if $c \geq 2$, then the expected cost to Alice is at least as much as the expected cost to a potentially faulty/spoofed Bob. If Bob happens to be faulty/spoofed, then the cost to him for an Ack Failure is less than the expected cost to Alice since a faulty/spoofed Bob will simply not listen in the Send Phase; as discussed above, we must avoid this since it admits a draining attack against Alice. Therefore, we have $1 < c < 2$. For Case 2, since Bob is guaranteed to be correct, the acceptable range is $1 < c \leq 2$.

Lemma 1. *Consider a non-send-blocking round of 3-PLAYER SCENARIO PROTOCOL. The probability that a correct Bob does not receive the message from Alice is less than e^{-2} .*

Proof. Let $s = 2^{ci}$ be the number of slots in the Send Phase. Let p_A be the probability that Alice sends in a particular slot. Let p_B be the probability that Bob listens in a particular slot. Let $X_j = 1$ if the message is not delivered from Alice to Bob in the j^{th} slot. Then $Pr[m \text{ is not delivered in the Send Phase}] = Pr[X_1 X_2 \cdots X_s = 1] = Pr[X_s = 1 \mid X_1 X_2 \cdots X_{s-1} = 1] \cdot \prod_{i=1}^{s-1} Pr[X_i = 1]$. Let $q_j = 1$ if Carol does not block in slot j ; otherwise, let $q_j = 0$. The value of q_j can be selected arbitrarily by Carol. Then $Pr[X_i = 1 \mid X_1 X_2 \cdots X_{i-1} = 1] = 1 - p_A p_B q_i$ and substituting for each conditional probability, we have $Pr[X_1 X_2 \cdots X_s = 1] = (1 - p_A p_B q_1) \cdots (1 -$

$p_{APB}q_s) = \prod_{j=1}^s (1 - p_{APB}q_j) \leq e^{-p_{APB} \sum_{j=1}^s q_j} < e^{-2}$ since $p_{APB} \sum_{j=1}^s q_j > (2/2^i)(2/2^{(c-1)^i})(s/2) = (2/2^i)(2/2^{(c-1)^i})(2^{ci}/2) = 2$ since the round is not send-blocking and so Carol blocks less than $s/2$ slots. \square

Note that Lemma 1 handles adaptive (but not reactive) adversaries. A simple but critical feature of tolerating adaptive adversaries is that the probability that a player is active in one slot is independent from the probability that the player is active in another slot. Therefore, knowing that a player was active for k slots in the past conveys no information about future activity. Believing otherwise is the trap of the well-known ‘‘Gambler’s Fallacy’’ [149]. For reactive adversaries, we need only modify Lemma 1 as we do later.

Lemma 2. *Assume that Bob is correct and there are no send-blocking rounds and no ack-blocking rounds. Then, the expected cost of each player is $O(S + L) = O(1)$.*

Proof. Using Lemma 1, the expected cost to Alice is at most $\sum_{i=2}^{\infty} e^{-2(i-2)} \cdot (2 \cdot 2^{(c-1)^i} \cdot S + 4 \cdot L) \leq \sum_{i=2}^{\infty} (e^{5-i} \cdot S + e^{2-2i} \cdot 4 \cdot L) = (e^5 \cdot S \cdot \sum_{i=2}^{\infty} e^{-i}) + (e^2 \cdot 4 \cdot L \cdot \sum_{i=2}^{\infty} e^{-2i}) = O(S + L) = O(1)$. Similarly, the expected cost to Bob is at most $\sum_{i=2}^{\infty} e^{-2(i-2)} \cdot (2^{i+1} \cdot L + 2^i \cdot S) \leq \sum_{i=2}^{\infty} (e^{5-i} \cdot L + e^{4-i} \cdot S) = O(S + L) = O(1)$ since S and L are constants. \square

Now consider when attacks may occur in the Ack Phase:

Lemma 3. *Assume that Bob has received m by round i and that round i is non-ack-blocking. Then the probability that Alice retransmits m in round $i + 1$ is less than e^{-2} .*

Proof. Let $s = 2^i$ be the number of slots in the Ack Phase and let $p = 4/2^i$ be the probability that Alice listens in a slot. For slot j , define X_j such that $X_j = 1$ if Alice does not terminate. Then $Pr[\text{Alice retransmits } m \text{ in round } i + 1] = Pr[X_1 X_2 \cdots X_s = 1]$. Let $q_j = 1$ if Carol does not block in slot j ; otherwise, let $q_j = 0$. The q_j values are determined arbitrarily by Carol. Since Alice terminates if and only if she listens and does not detect any activity, then $Pr[X_j = 1] = (1 - pq_j)$. Therefore, $Pr[X_1 X_2 \cdots X_s = 1] \leq e^{-p \sum_{j=1}^s q_j} < e^{-2}$. \square

Lemma 4. *Assume there is at least one send-blocking round. Then, the expected cost to Alice is $O(B^{(c-1)/c} + B^{(c-1)})$ and the expected cost to a correct Bob is $O(B^{1/c})$.*

Proof. We consider Case 1 and Case 2 with regards to Bob, discussed in Section 3.1.1. Let $i \geq 2$ be the last round which is send-blocking. Let $j \geq i$ be the last round which is ack-blocking; if no such ack-blocking round exists, then assume $j = 0$. In Case 1, the total cost to Carol is $B = \Omega(2^{ci} \cdot J + 2^j \cdot J) = \Omega(2^{ci} + 2^j)$ since J is a constant. In Case 2, only send-blocking occurs and so $B = \Omega(2^{ci} \cdot J)$.

Alice: We first calculate the expected cost to Alice prior to successfully transmitting m . In round i , Carol blocks the channel for at least $2^{ci}/2$ slots. Using Lemma 1, the expected cost to Alice prior to m being delivered is $O(2^{(c-1)i} \cdot S + 4 \cdot L) + \sum_{k=1}^{\infty} e^{-2(k-1)} \cdot (2 \cdot 2^{(c-1)(i+k)} \cdot S + 4 \cdot L) = O(2^{(c-1)i} \cdot S + L) = O(2^{(c-1)i})$ by the bounds on c and given that S and L are constants; note, this is the total cost to Alice for Case 2.

Now, using Lemma 3, we calculate the expected cost to Alice after delivery; this addresses ack-blocking rounds possible only in Case 1. By assumption, the last ack-blocking round occurs in round j and therefore Alice's expected cost is $O(2^{(c-1)j} \cdot S + 4 \cdot L) + \sum_{k=1}^{\infty} e^{-2(k-1)} \cdot (2 \cdot 2^{(c-1)(j+k)} \cdot S + 4 \cdot L) = O(2^{(c-1)j} \cdot S + L)$ by the bounds on c . Therefore, the total expected cost to Alice is $O(2^{(c-1)i} \cdot S + 2^{(c-1)j} \cdot S + L) = O(2^{(c-1)i} + 2^{(c-1)j})$. Since $B = \Omega(2^{ci} + 2^j)$, this cost as a function of B is $O(B^{(c-1)/c} + B^{(c-1)})$.

Bob: Finally, assume Bob is correct. Using Lemma 1, Bob's expected cost prior to receiving m is $O(2^{i+1} \cdot L + 2^i \cdot S) + \sum_{k=1}^{\infty} e^{-2(k-1)} \cdot (2 \cdot 2^{i+k} \cdot L + 2^{i+k} \cdot S) = O(2^i \cdot L + 2^i \cdot S) = O(2^i)$ since S and L are constants. Thus, the expected cost for Bob as a function of B is $O(B^{1/c})$. \square

We now give the proof for Theorem 1 stated in Section 3.1.3:

Proof of Theorem 1: In Case 1, Lemma 4 tells us that the expected cost to Alice and Bob in terms of B is $O(B^{(c-1)/c} + B^{(c-1)})$ and $O(B^{1/c})$, respectively. Therefore, the exponents of interest which control the cost to each player are $(c-1)/c$, $c-1$, and $1/c$. The value of c that should be chosen must minimize $\max\{(c-1)/c, c-1, 1/c\}$ since we are interested in fair protocols. Given that $1 < c < 2$, we have $1/c > (c-1)/c$. Therefore, we solve for c in $c-1 = 1/c$,

this gives $c = (1 + \sqrt{5})/2$ which is the golden ratio. By Lemma 2 and the above argument, the expected cost to each player is $O(B^{\varphi-1} + 1)$. In Case 2, Lemma 4 tells us that Alice’s expected cost in terms of B is $O(B^{(c-1)/c})$ the exponents of interest are simply $(c - 1)/c$ and $1/c$; minimizing them yields $c = 2$. Therefore, the cost to each player is $O(B^{1/2} + 1)$.

Finally, define latency to be the number of slots that occur prior to termination by both correct players. Consider how many non-send-blocking or non-ack-blocking rounds *either* player may endure before terminating successfully; let X denote the random variable for this number of rounds. Then, $E[X] \leq 1 \cdot (1 - e^{-2}) + 2 \cdot e^{-2}(1 - e^{-2}) + 3 \cdot e^{-4}(1 - e^{-2}) + \dots = \sum_{i=1}^{\infty} i e^{2(1-i)}(1 - e^{-2}) = (1 - e^{-2})e^2 \sum_{i=1}^{\infty} i(e^{-2})^i$ by Lemmas 1 or 3. Therefore, $E[X] \leq 1/(1 - e^{-2}) = O(1)$ which translates into $O(1)$ time slots consumed by non-send or non-ack-blocking rounds. Now consider the send-or-ack-blocking rounds; note that Carol is limited to at most $\lg(2B) + O(1)$ such rounds which translates to $O(B^\varphi)$ time slots. Therefore, regardless of how Carol blocks, the expected number of time slots prior to successful termination is $O(B^\varphi)$. \square

3.2.2 Tolerating a Reactive Adversary

Consider a reactive adversary Carol who can detect channel activity without cost, and then block the channel; this ability is possible in WSNs (see Section 3.3.1). In our 3-Player Scenario, Carol can now detect that m is being sent in the Send Phase and block it without fail. To address this powerful adversary, we consider the case where critical data, m , and more often, non-critical data m' , is sent over the channel by other participants in addition to Alice and Bob. Carol can detect the traffic; however, she cannot discern whether it is m or m' without listening to a portion of the communication (such as packet header information).

In a slot where channel activity is detected, even if Carol listens for a portion of the message, she incurs a substantial cost. Therefore, the cost to Carol is proportional to the number of messages to which she listens. Importantly, in the presence of m' , Carol’s ability to detect traffic for free is unhelpful since m' provides “camouflage” for m . Certainly Carol may block *all active slots* to prevent transmission of m ; however, this is no different than blocking *all slots* in our original 3-Player Scenario.

As an extreme example, assume that *all* slots in the Send Phase are used either by Alice to send m (as per our protocol) or Dave, whose transmissions of m' do not interest Carol, and the probability that a slot is used by Dave is higher. Then detecting channel activity does not help Carol decide on whether to block; all slots are used. Regardless of how she decides to act, Carol can do no better than picking slots independent of whether she detects channel activity. In other words, channel activity is no longer useful in informing Carol's decisions about whether to block.

But assuming *all* slots are active is problematic: (1) How is this guaranteed or coordinated? (2) Doesn't this much background traffic interfere with Bob's ability to receive from Alice? Instead, assume that other network traffic occurs such that Carol will always detect traffic on at least a constant fraction of slots in the Send Phase. Note that this does not help her block transmissions by Alice since she does not know the total amount of traffic that she will detect. Now, not all slots will necessarily be active. Upon detecting traffic, can Carol listen to a portion of the message to discover if it is m or m' and then decide on whether to block? Yes, but this is roughly as expensive as simply blocking outright. So again, detecting channel activity does not inform Carol's decisions. This is the idea behind our analysis.

This corresponds to situations where communication occurs steadily between many participants or via several distributed applications, and Carol wishes to target only a critical few. If m and m' are sent over the channel in the same slot, the two messages collide and Bob receives neither. Define a slot as *active* if either m or m' is sent in that slot. For this result only, redefine a send-blocking round as one where Carol listens or blocks for at least a $1/3$ -fraction of the *active* slots; otherwise, it is a *non-send-blocking round*. We provide a result analogous to Lemma 1.

Lemma 5. *Let Carol be an adaptive and reactive adversary. Then, in a non-send-blocking round of the 3-PLAYER SCENARIO PROTOCOL, the probability that Bob does not receive m from Alice is at most e^{-2} .*

Proof. Let $x = 2^{ci}$ be the number of slots in the Send Phase. Consider the set of slots used by all participants other than Alice. We assume these participants pick their slots at random to send, so that for any slot the probability is $2/3$ that the slot is chosen by at least one of them. Since we assume these messages m' are sent independently at random, then Chernoff

bounds imply that, with high probability (i.e., $1 - 1/x^{c'}$ for constants c', ϵ and sufficiently large x) the number of slots y during which m' is sent is greater than $(2x/3)(1 - \epsilon)$ where x is the total number of slots in a phase. In the same way, assume the number of slots in which Alice sends is at least $a = (1 - \delta)xp_A = (1 - \delta)2^{(c-1)i+1}$ with probability $1 - 1/x^{c''}$ for a constant δ, c'' and sufficiently large x . The number of active slots sent by Alice or other participants is clearly at least y .

By definition of a non-send-blocking round, Carol listens to or blocks less than $x/3$ (active) slots. As Carol has no information about the source of a message sent in an active slot until she listens to it, her choice is independent of the source of the message. Given a slot that Alice sends on, there is at least a $1 - (x/3)/y$ chance it will not be listened to or blocked by Carol. The probability that this slot will not be used by another participant is $1/3$ and the probability that Bob will listen to the slot is p_B . Hence the probability of a successful transmission from Alice to Bob on a slot which Alice sends on is at least $p = (1 - x/(3y))(1/3)p_B = (1 - 1/(2(1 - \epsilon)))(1/3)p_B \geq (1/3 - (1 + \delta)/6)p_B$ for sufficiently large x (that reduces the size of δ) when $y > (1 - \epsilon)(2x/3)$. Therefore, we can write $p \geq (1/12)p_B$. The probability that all messages that Alice sends fail to be delivered is at most $(1 - p_B/12)^a + 2/x^{c''}$ where the last term is the probability of the bad event that y or a is small and $c'' > 0$ is a constant. Redefine $p_B = 24/((1 - \delta)2^{(c-1)i})$ where the value 24 is there to simply off-set the additive $2/x^{c''}$ term. Note that this constant factor increase in the listening probability does not change our asymptotic results and our analysis in Section 3.2.1 proceeds almost identically. Therefore, we then have $(1 - p_B/12)^a + 2/x^{c''} \leq e^{-2}$. \square

The 3-PLAYER SCENARIO PROTOCOL can be modified so that the initial value of i is large enough to render the error arising from the use of Chernoff bounds sufficiently small; we omit these details. Also, the required level of channel traffic detected by Carol is flexible and different values can be accommodated if the players' probabilities for sending and listening are modified appropriately in the 3-PLAYER SCENARIO PROTOCOL; our results hold asymptotically. Finally, we emphasize that Lemma 3 does not require modification. Carol cannot decide to block only when Alice is listening since detecting when a node is listening is impossible. Alternately, Carol cannot silence a req through (reactive) blocking since this is still interpreted as a retransmission request. Using Lemma 5, Theorem 1 follows as before.

Finally, we note that the conclusion of our argument aligns with claims put forth in empirical results on reactive jamming in WSNs; that is, such behaviour does not necessarily result in a more energy-efficient attack because the adversary must still be listening to the channel for broadcasts prior to committing itself to their disruption [165].

3.2.3 On Latency and Lower Bounds

In Chapter 1, we mentioned the *Bad Santa* problem which is described as follows. A child is presented with K boxes, one after another. When presented with each box, the child must immediately decide whether or not to open it. If the child does not open a box, it can never be revisited. Half the boxes have presents in them, but the decision as to which boxes have presents is made by an adversarial Santa who wants the child to open as many empty boxes as possible. The goal is for the child to obtain a present *with probability 1*, while opening the smallest expected number of boxes. In [87, 88], the authors prove a lower bound of $\Omega(K^{0.5})$ on the expected number of opened boxes.

Theorem 5. *Any algorithm that solves the 3-Player Scenario with $o(B^{0.5})$ cost to Bob must have a latency exceeding $2B$.*

Proof. A lower bound for the 3-Player Scenario is complicated by the possibility that the strategies of Alice and Bob may adapt over time; for example, they may change depending on how Carol blocks. To address this, we assume a more powerful Bob. Specifically, assume that communication of m occurs if Bob is able to find an unblocked time slot in which to listen *or* to send. Furthermore, assume Bob can tell when he has found such a slot once he listens or sends in that slot. Therefore, such a Bob is at least as powerful as the Bob in the 3-Player Scenario.

Now, if Carol has a budget of size B , we ask: Does Bob have a strategy with $o(B^{0.5})$ expected active slots such that, with probability 1, he finds at least one unblocked slot within $2B$ slots? Assume that such a strategy exists and consider the Bad Santa problem on $2B$ boxes. Using Bob's strategy, the child is guaranteed to obtain a present with probability 1 while opening $o(B^{0.5})$ boxes in expectation. However, this contradicts the $\Omega(B^{0.5})$ lower bound result in [87] and the result follows. \square

This result illustrates a relationship between the Bad Santa problem and the 3-Player Scenario, and it provides some insight into why our protocol has a worst-case latency of $\omega(B)$ slots.

3.3 Application 1: Jamming Resistance in Wireless Sensor Networks

The shared wireless medium of sensor networks renders them vulnerable to jamming attacks [154]. A jamming attack occurs when an attacker transmits noise at high energy, possibly concurrently with a (legitimate) transmission, such that communication is disrupted within the area of interference. Consequently, this behaviour threatens the availability of sensor networks [162].

3.3.1 Rationale for the 3-Player Scenario Involving WSN Devices

Wireless network cards offer states such as *sleep*, *receive (or listen)* and *transmit (or send)*. While the sleep state requires negligible power, the cost of the send and listen states are roughly equivalent and dominate the operating cost of a device. For example, the send and listen costs for the popular Telos motes are 38mW and 35mW, respectively (note $S \approx L$) and the sleep state cost is $15\mu\text{W}$ [124]; therefore, the cost of the send/listen state is more than a factor of 2000 greater and the sleep state cost is negligible. Disruption may not require jamming an entire slot so we set $J < S$ and assume a small m such that J and S are within a constant factor of each other; larger messages can be sent piecewise. In our protocols, we account for both send and receive costs. Throughout, when a node is not active, we assume it is in the energy-efficient sleep state.

Slots: There is a single channel and a time division multiple access (TDMA)-like medium access control (MAC) protocol; that is, a time-slotted network. For example, the well-known LEACH [72] protocol is TDMA-based. For simplicity, a *global* broadcast schedule is assumed; however, this is likely avoidable if nodes maintain multiple schedules as with S-MAC [167]. Even then, global scheduling has been demonstrated by experimental work in [97] and secure synchronization has been shown [55].

A blocked slot occurs when Carol jams. Clear channel assessment (CCA), which subsumes carrier sensing, is a common feature on devices for detecting such events [126] and practical under the IEEE 802.11 standard [44]. *Collisions are only detectable by the receiver* [162]. When a collision occurs, a correct node discards any received data. We assume that the absence of channel activity cannot be forged by the adversary; this aligns with the empirical work by Niculescu [114] who shows that channel interference increases linearly with the combined rate of the sources. Finally, we also note that several theoretical models feature collision detection (see [4, 11, 24, 62, 130]).

On Reactive Adversaries: CCA is performed via the radio chip using the *received signal strength indicator* (RSSI) [145]. If the RSSI value is below a clear channel threshold, then the channel is assumed to be clear [15]. Such detection consumes on the order of 10^{-6} W which is three orders of magnitude smaller than the send/listen costs; therefore, Carol can detect activity (but not message content) at essentially zero cost. Listening to even a small portion of a message costs on the order of milliwatts and our argument from Section 3.2.2 now applies.

Cryptographic Authentication: We assume that messages can be authenticated. Therefore, Carol cannot spoof Alice; however, Bob’s `req` can essentially be spoofed by an Ack-Failure (as discussed in Section 3.2) which, along with jamming, makes the problem non-trivial. Several results show how light-weight cryptographic authentication can be implemented in sensor networks [83, 94, 100, 154, 161]; therefore, it is important to consider its impact as we do here. However, the adversary may capture a limited number of players (such as Bob); these players are said to suffer a Byzantine fault and are controlled by the adversary [154, 162]. Given this attack, we emphasize that, while we assume a shared key to achieve authentication, *attempts to share a secret send/listen schedule between Alice and Bob allows Carol to manipulate players in ways that are problematic*. This is discussed further in Section 3.3.3.

3.3.2 Local Broadcast and Guaranteed Latency

Our protocol LOCAL BROADCAST handles the general single-hop broadcast situation where Alice sends m to a set of n neighboring receivers within her transmission range. At first glance, this seems achievable by having each

<p>LOCAL BROADCAST($m, \text{Alice}, R_{\text{Alice}}$) for round $i \geq \lg(4 \ln n)$</p> <p><i>Probabilistic Send Phase:</i> For each of the 2^{φ^i} slots do</p> <ul style="list-style-type: none"> • Alice sends m with probability $\frac{3 \ln n}{2^i}$. • Each receiver that has not terminated listens with probability $\frac{2}{2^{(\varphi-1)^i}}$. <p><i>Deterministic Send Phase:</i> For each of the $2^{(\varphi-1)^{i+1}}$ slots do</p> <ul style="list-style-type: none"> • Alice sends m. • Each receiver that has not terminated listens. <p>Any receiver that receives m terminates the protocol.</p> <p><i>Probabilistic Ack Phase:</i> For each of the 2^i slots do</p> <ul style="list-style-type: none"> • Each receiver that has not terminated sends a req message. • Alice listens with probability $\frac{4 \ln n}{2^i}$. <p><i>Deterministic Ack Phase:</i> For each of the $2^{(\varphi-1)^{i+1}}$ slots do</p> <ul style="list-style-type: none"> • Each receiver that has not received m sends a req message. • Alice listens. <p>If Alice listened in either a Probabilistic Ack Phase or a Deterministic Ack Phase and detected no req message or collision then she terminates the algorithm.</p>

Figure 3.2: Pseudocode for LOCAL BROADCAST.

receiver execute an instance of 3PSP with Alice. However, the expected active time for Alice is an $\Omega(n)$ -factor larger than any correct receiver; thus, this is unfair. Furthermore, this protocol has poor latency. Here, we give a fast protocol that is both fair and favourable up to small polylogarithmic factors.

Our pseudocode is given in Figure 3.2 which is valid for $n \geq 2$. The probabilities for sending and listening are modified and there are two more phases (the Deterministic Send and Deterministic Ack Phases) where players act deterministically. Note that **req** messages can collide in the Probabilistic Ack Phase and will certainly collide in the Deterministic Ack Phase. This is correct as such a collision is due to either jamming or multiple receivers (correct or faulty) requesting a retransmission; this is fine and Alice will resend. LOCAL BROADCAST takes in as arguments the message m , the sender (Alice) and the set of receivers R_{Alice} . If the adversary jams, then none of the correct receivers receive m in that slot. We now prove the properties of LOCAL BROADCAST.

Lemma 6. *Consider a non-send-blocking round. The probability that at least*

one correct receiver does not receive the message from Alice is less than $1/n^2$.

Proof. Let s be the number of slots in the Probabilistic Send Phase of round i . Let $p_A = 3 \ln n / 2^i$ be the probability that Alice transmits in a particular slot. Let $p_b = 2/2^{(\varphi-1)i}$ be the probability that a particular correct receiver b listens in a particular slot. Let $X_j = 1$ if the message is not transmitted from Alice to receiver b in the j^{th} slot. Then $\Pr[m \text{ is not successfully transmitted to the } b \text{ during the Probabilistic Send Phase}] = \Pr[X_1 X_2 \cdots X_s = 1] = \Pr[X_s = 1 \mid X_1 X_2 \cdots X_{s-1} = 1] \cdot \Pr[X_1 X_2 \cdots X_{s-1} = 1]$. Let $q_j = 1$ if the adversary does not jam given $X_1 X_2 \cdots X_{i-1}$; otherwise, let $q_j = 0$. The value of q_j can be selected arbitrarily by the adversary. Then $\Pr[X_i = 1 \mid X_1 \cdots X_{i-1} = 1] = 1 - p_{AP_b} q_i = 1 - (6 \ln n / 2^{\varphi i}) q_j$. Then we have $\Pr[X_1 X_2 \cdots X_s = 1] = (1 - p_{AP_b} q_1) \cdots (1 - p_{AP_b} q_s) \leq \prod_{j=1}^s (1 - p_{AP_b} q_j) \leq e^{-p_{AP_b} \sum_{j=1}^s q_j} < 1/n^3$ since $p_{AP_b} \sum q_j > (6 \ln n / 2^{\varphi i}) \cdot (2^{\varphi i} / 2) = 3 \ln n$ given that this is a non-send-blocking round. Taking a union bound, the probability that at least one correct receiver has not received m is less than n^{-2} . \square

A notable difference between Lemma 6 and the previous Lemma 1 is that here we want to bound the probability that *any* correct node has not received m . This requires a union bound which necessitates the $O(\log n)$ factor in our new analysis. A slightly tighter analysis may be possible; however, it should not change things asymptotically. We also note Lemma 6 can be modified to handle a reactive adversary in the same way as done for 3-PLAYER SCENARIO PROTOCOL; we omit the details.

Lemma 7. *Assume that by round i all correct receivers have heard the message m . Assume that round i is non-ack-blocking. Then the probability that Alice retransmits the message in round $i + 1$ is less than $1/n^2$.*

Proof. This is computed similarly to the proof of Lemma 6. Let s be the number of slots in the Probabilistic Ack Phase and let $p = 4/2^i$ be the probability that Alice listens in a slot. For slot j , define X_j such that $X_j = 1$ if Alice does not terminate. Then $\Pr[\text{Alice retransmits } m \text{ in round } i + 1] = \Pr[X_1 X_2 \cdots X_s = 1]$. Let $q_j = 1$ if the adversary does not jam given $X_1 X_2 \cdots X_{i-1}$; otherwise, let $q_j = 0$. The q_j values are determined arbitrarily

by the adversary who controls the faulty receivers. Since Alice terminates if and only if it listens and does not detect any activity, then $Pr[X_j = 1] = (1 - pq_j)$. Therefore, $Pr[X_1 X_2 \cdots X_s = 1] \leq e^{-p \sum_{j=1}^s q_j} < n^{-2}$ since $p \sum_{j=1}^s q_j > (4 \ln n / 2^i)(2^i / 2) = 2 \ln n$ given that this is a non-ack-blocking round. \square

Lemma 8. *Assume all receivers are correct and there are no send-blocking or ack-blocking rounds. Then the expected cost to Alice is $O(\ln^\varphi n)$ and the expected cost to any correct receiver is $O(\ln n)$.*

Proof. Let $d = \lg(4 \ln n)$ and $n \geq 3$. Using Lemma 6, the expected cost to Alice is at most:

$$\begin{aligned} & \sum_{i=d}^{\infty} n^{-2(i-d)} \cdot (2^{(\varphi-1)i} \cdot 3 \ln n + 2^{(\varphi-1)i+1} + 4 + 2^{(\varphi-1)i+1}) \\ &= O(\ln^\varphi n) + O\left(\ln n \cdot \sum_{k=1}^{\infty} \left(\frac{2^{(\varphi-1)}}{n^2}\right)^k\right) \\ &= O(\ln^\varphi n) \text{ by the geometric series.} \end{aligned}$$

Similarly, using Lemma 7, the expected cost to each receiver is at most:

$$\begin{aligned} & \sum_{i=d}^{\infty} n^{-2(i-d)} \cdot (2^{i+1} + 2^{(\varphi-1)i+1} + 2^i + 2^{(\varphi-1)i+1}) \\ &= O(\ln n) + O\left(\sum_{k=1}^{\infty} \left(\frac{2}{n^2}\right)^k\right) \\ &= O(\ln n) \text{ by the geometric series.} \end{aligned}$$

\square

Lemma 9. *Assume there is at least one send-blocking round. The expected cost to Alice is $O(B^{\varphi-1} \ln n + \ln^\varphi n)$ and the expected cost to any correct receiver is $O(B^{\varphi-1} + \ln n)$.*

Proof. Let $i \geq \lceil \lg(4 \ln n) \rceil$ be the last round which is send-blocking and let j be the last round which is ack-blocking, $j \geq i$; if no such ack-blocking round exists, then $j = 0$. Then the cost to the adversary is $B = \Omega(2^{\varphi i} + 2^j)$.

Alice: Using Lemma 7, the expected cost to Alice prior to successfully terminating is $O(2^{(\varphi-1)i} \ln n) + \sum_{k=1}^{\infty} n^{-2(k-1)} \cdot O(2^{(\varphi-1)(j+k)} \ln n) = O(2^{(\varphi-1)i} \ln n + 2^{(\varphi-1)j} \ln n)$. Therefore, in terms of B , the cost to Alice is $O(B^{\varphi-1} \ln n)$ and by Lemma 8, Alice's total expected cost is $O(B^{\varphi-1} \ln n + \ln^{\varphi} n)$.

Correct Receivers: In the worst case, all rounds up to i have been send-blocking, in which case the expected cost to each correct receiver up to the end of round $i+1$ is $O(2^i)$. Therefore, in terms of B , and using Lemma 8, the cost to each correct receiver is $O(B^{\varphi-1} + \ln n)$ noting that $1/\varphi = \varphi - 1$. \square

Lemma 10. *Alice and all correct receivers terminate LOCAL BROADCAST in $25 \cdot (B + 2^{\varphi-1} \ln^{\varphi-1} n)^{\varphi+1}$ time slots.*

Proof. The deterministic phases play a key role in establishing the bound on latency. If the adversary is not active for all slots in the deterministic Send Phase, then all correct receivers obtain m . Once all correct receivers terminate, the adversary must be active in all slots of the deterministic Ack Phase in order to prevent Alice from terminating. Therefore, prior to successful termination of all correct players (including Alice), the adversary is active for at least $2^{(\varphi-1)i+1}$ slots per round i in Epochs 2 and 4. For $d = \lg(4 \ln n)$, we seek the number of rounds ρ such that $\sum_{i=d}^{\rho} 2^{(\varphi-1)i+1} \geq B$ which yields that $\rho \geq \varphi \lg(B + 2^{\varphi-1} \ln^{\varphi-1} n)$ rounds suffices to exhaust the adversary (we are not being exact). Each round i has at most $4 \cdot 2^{\varphi i+1}$ slots so ρ rounds equal at most $25 \cdot (B + 2^{\varphi-1} \ln^{\varphi-1} n)^{\varphi+1}$ slots. \square

The value n is the number of devices within the broadcast range of Alice. Throughout, we have assumed that n is known *a priori*. There has been work done on node discovery in the presence of jamming (see Meier *et al.* [104]) and we assume that similar techniques can be used to obtain the value n . For a determined adversary, we expect $B > n$; that is, for an adversary intent on preventing communication, the number of time slots jammed will likely exceed the number of neighbors. Therefore, $B \gg \ln^{\varphi+1} n$. In this case (actually for $B \geq \ln^{\varphi-1} n$), the latency is $O(B^{\varphi+1})$ and, noting that Carol can prevent transmission for at least B slots, this is within an $O(B^{\varphi})$ -factor of the optimal latency. By this and Lemmas 8, 9, and 10, Theorem 2 now follows.

3.3.3 Why a Shared Schedule is Problematic

In our WSN application, we assume that messages from Alice can be authenticated using light-weight cryptographic techniques. Given this, we consider: might Alice and Bob (or even more players) also share a secret schedule? This would reduce the costs in Theorem 1 due to the Send Phase where neither player knows if the other is active with any certainty.

Unfortunately, such a schedule becomes known to the adversary if a player suffers a Byzantine fault and this causes problems in more general scenarios. For instance, consider the simple extension of Alice and two receivers. In our local broadcast problem, which is a key subroutine for our reliable broadcast protocol, Alice broadcasts to its two neighboring receivers concurrently in order to be fair. Therefore, both receivers must know when Alice transmits in the Send Phase. By corrupting one receiver, this schedule becomes known to the adversary who can then block transmissions by Alice perfectly and easily prevent the other receiver from receiving m . Clearly, this attack extends to the case where there are n receivers and Alice wants to achieve a local broadcast.

Other problems arise in a multi-hop scenario. For example, in our reliable broadcast protocol, each node listens to many different senders. A faulty receiver can interfere with many more senders by acting in the same manner as above for each of these senders. Therefore, by purposely avoiding a pre-set shared schedule, our use of randomness allows us to foil such attempts by the adversary.

3.3.4 Jamming-Resistant Reliable Broadcast: Mitigating the Listening Cost Disadvantage

Reliable broadcast has been extensively studied in the multi-hop grid model [21–23, 88, 90], particularly with a jamming adversary [4, 20, 24]. Reliable broadcast is possible when t Byzantine nodes can each jam at most n_c transmissions [24]. Unfortunately, the protocol of [24], and the improvement by [20], requires that *correct nodes possess more energy than the Byzantine nodes*. In particular, while the sending costs are improved in [20], both [20, 24] allow the adversary to force a correct node to *listen* for $\Omega(t \cdot n_c)$ slots where n_c is the number of times each node can cause a message collision (listening

costs in [4] are similar). In contrast, each Byzantine node is active for n_c slots. This $\Omega(t)$ -factor advantage affords the adversary a DDoS attack since these previous protocols are consistently unfavourable.

Reliable Broadcast in the Grid

We reiterate the grid model: each node $p(x, y)$ is situated at (x, y) in a grid. The dealer d (who is correct) is located at $(0, 0)$ and seeks to propagate m to all correct nodes in the network. When a node p sends a message, all listening nodes within L_∞ distance r (i.e. the $(2r + 1) \times (2r + 1)$ square centered about p) receive the message; this *neighborhood* is denoted by $N(p)$. Analogous results hold for the Euclidean metric (see [22]). There are $t < (r/2)(2r + 1)$ Byzantine nodes in any neighborhood. Unlike the single-hop case, here the amount of jamming in a neighborhood is upper bounded by B_0 and known. This is required in [20, 24] and a similar assumption is made in [11, 130]. B_0 represents the number of times a Byzantine node can deviate from the global schedule within some time frame in a neighborhood before being identified and subjected to defensive techniques (see [162]). Not exceeding B_0 in each time frame allows the adversary to attack throughout the lifetime of the network and we pessimistically assume that B_0 is large so that the adversary may inflict sustained attacks.

There is a global broadcast schedule (obeyed by the correct nodes) that assigns each node a slot for broadcasting; the ordering is always the same but the actual specification is unimportant (see [90] for an example). A *cycle* is defined as on full pass through a global broadcast schedule —we call these *transmit slots* —plus an additional n slots that we call *response slots*; we expand on this later.

Overview of the Protocol: The pseudocode is in Figure 3.3. Our protocol synchronizes the timing of nodes for sending and listening. While this synchronization is not mathematically challenging, a full description yields an unreadable protocol. *For ease of exposition, our treatment addresses each node q in $C = \{q(x, y) \mid -r \leq x \leq r \wedge y \geq 0\}$; that is, a corridor of width $2r + 1$ moving up from d .* Traversing the x -coordinates is nearly identical and the grid can be covered piecewise by these two types of corridors.

For each node p , define $A_p = \{q(u, v) \mid (a - r) \leq u \leq (a + z) \text{ and } (b + 1) \leq v \leq (b + r)\}$, $B_p = \{q(u, v) \mid (a + z + 1) \leq u \leq (a + r) \text{ and } (b + 1) \leq v \leq (b + r)\}$

DoS-Resistant Reliable Broadcast

- 1: Starting in cycle 1, and ending no later than cycle $D = 25 \cdot (B_0 + 2^{\varphi-1} \ln^{\varphi-1} n)^{\varphi+1}$, node d executes LOCAL BROADCAST(m, d, R_d) and each node $i \in R_d$ commits to the first value it receives from d .

As in [22], $p(x, y)$ commits to m when, through Steps 4 and 5, it receives $t + 1$ COMMIT(q, m) or HEARD(q_2, q_1, m) from node-disjoint paths all lying within a single $(2r + 1) \times (2r + 1)$ area; our analysis shows this occurs in cycle $2yD - 1$. The following step is executed by each node p :

- 2: Starting in cycle $2yD$, and ending no later than cycle $(2y + 1)D - 1$, node $p(x, y)$ performs LOCAL BROADCAST(COMMIT(p, m), p, R_p).

The following steps are executed by each node excluding those nodes in $N(d)$:

- 3: **for** $i = 0$ to $r - 1$ **do**
- 4: Starting in cycle $2(y - r + i)D$, and ending no later than cycle $2(y - r + i)D + D - 1$, node $p(x, y)$ *listens* for COMMIT messages by executing LOCAL BROADCAST(COMMIT(q, m), $q(x', y')$, R_q) with each node in row $y' = y - r + i$ in C and where $p \in R_q$.
- 5: Starting in cycle $2(y - r + i)D + D$, and ending no later than cycle $2(y - r + i)D + 2D - 1$, node $p(x, y)$ *listens* for HEARD messages by executing LOCAL BROADCAST(HEARD(q_2, q_1, m), q_2, R_{q_2}) with each node $q_2 \in B'_p$ in row $y + i$ and where $p \in R_{q_2}$.
- 6: Starting in cycle $2(y - r)D + D$, and ending no later than cycle $2(y - r)D + 2D - 1$, node q_2 *sends* a HEARD message by executing LOCAL BROADCAST(HEARD(q_2, q_1, m), q_2, R_{q_2}) where q_1, q_2 are sister nodes.

Figure 3.3: Pseudocode for DOS-RESISTANT RELIABLE BROADCAST.

and $B'_p = \{q'(u', v') \mid (a + z + 1 - r) \leq u' \leq (a) \text{ and } (b + r + 1) \leq v' \leq (b + 2r)\}$ for $0 \leq z \leq r$. The set B'_p is obtained from B_p by shifting left by r units and up by r units; under this 1-to-1 translation, $q_1 \in B_p$ and $q_2 \in B'_p$ are sister nodes. The reader is referred to [22] or [88] for a more in-depth discussion of these sets.

While a full presentation is somewhat tedious, the main idea is that where a node would have broadcasted a message to a group of nodes in

the protocol of Bhandari and Vaidya [22], we now use LOCAL BROADCAST to communicate a message to that group of nodes. In terms of the messages themselves, node q issues a COMMIT(q, m) message if q has committed to m . Node q_2 sends HEARD(q_2, q_1, m) if q_2 has received a message COMMIT(q_1, m). As in [22], p commits to m when it receives $t + 1$ COMMIT(q, m) or HEARD(q_2, q_1, m) from node-disjoint paths all lying within a single $(2r + 1) \times (2r + 1)$ area.

We now discuss how to move from slots in LOCAL BROADCAST to slots in a cycle. In general, the transmit slots in a cycle are used by a node p to transmit a HEARD or COMMIT message via LOCAL BROADCAST to a receiving set of nodes R_p , while the response slots in a cycle are used by nodes in R_p to send back req messages to p . Therefore, there are up to n transmit slots needed. For simplicity, we do not go into detail about how these are set up; we simply assume that nodes in R_p know which response slot to use.

In our p in our pseudocode, $R_p = N(p) \cap C$ and p executes LOCAL BROADCAST(m, p, R_p) in the context of the global broadcast schedule. By this, we mean that a slot in the Probabilistic Sending Phase of LOCAL BROADCAST corresponds to p 's transmit slot in some cycle, the next slot in that same Probabilistic Sending Phase corresponds to p 's transmit slot in the next cycle, and so on. The same thing happens with the Deterministic Sending Phase. In both cases, the response slots are unused. Then in the Probabilistic Ack Phase and Deterministic Ack Phase, the responses are used by each R_p set in the same fashion, where p now listens. By using a response slot, the nodes in R_p send back to p simultaneously as in LOCAL BROADCAST. Note that in the Probabilistic and Deterministic Ack phases, the transmit slots are now unused. Therefore, in each cycle, only the transmit slots or response slots, but not both, are used. For LOCAL BROADCAST running in at most D slots, executing LOCAL BROADCAST in the context of the global broadcast schedule requires at most D cycles. In Figure 3.3, we have $D = 25 \cdot (B_0 + 2^{\varphi-1} \ln^{\varphi-1} n)^{\varphi+1}$ in concordance with Lemma 10.

We include the detailed proof of completeness for Theorem 3 by showing that each correct node eventually commits to the correct value m sent by the dealer (the cost analysis is provided later). The following Lemma 11 proves the correctness of our protocol in the grid; we emphasize that our argument follows that of [22].

Lemma 11. *Assume for each node p , $t < (r/2)(2r + 1)$ nodes in $N(p)$ are*

*Byzantine and used by Carol to disrupt p 's communications for $\beta \leq B_0$ time slots. Let $C = \{\text{Nodes } q \text{ at } (x, y) \mid -r \leq x \leq r \wedge y \geq 0\}$ be a corridor of nodes in this network. Then, **DOS-RESISTANT RELIABLE BROADCAST** achieves reliable broadcast in C .*

Proof. In [22], it is shown that each node $p(x, y)$ can obtain m by majority filtering on messages from $2t+1$ node-disjoint paths contained within a single $(2r+1) \times (2r+1)$ area since at least $t+1$ will be m . Our correctness proof is similar; however, we argue along a corridor and show that nodes in the y^{th} row can commit to m by slot $2yD - 1$.

Base Case: Each node in $N(d)$ commits to the correct message m immediately upon hearing it directly from the dealer by cycle D . Therefore, clearly, every node $p(x, y) \in N(d)$ commits by cycle $2yD - 1$.

Induction Hypothesis: Let $-r \leq a \leq r$. If each correct node $p'(x', y') \in N(a, b)$ commits to m by cycle $2y'D - 1$, then each correct node $p(x, y) \in N(a, b+1) - N(a, b)$ commits to m in cycle $2yD - 1$.

Induction Step: We now show $2t+1$ connectedness within a single neighborhood and we argue simultaneously about the time required for p to hear messages along these disjoint paths. The node $p(x, y)$ lies in $N(a, b+1) - N(a, b)$ and can be considered to have location $(a - r + z, b + r + 1)$ where $0 \leq z \leq r$ (the case for $r + 1 \leq z \leq 2r$ follows by symmetry). We demonstrate that there exist $r(2r+1)$ node-disjoint paths $P_1, \dots, P_{r(2r+1)}$ all lying within the same neighborhood and that the synchronization prescribed by our protocol is correct:

One-Hop Paths: the set of nodes $A_p = \{q(u, v) \mid (a - r) \leq u \leq (a + z) \text{ and } (b + 1) \leq v \leq (b + r)\}$ lie in $N(a, b)$ and neighbor p . Therefore, there are $r(r + z + 1)$ paths of the form $q \rightarrow p$ where $q \in A_p$.

By their position relative to $p(x, y)$, each correct node $q(u, v) \in A_p$ is such that $v = y - r + c$ for some fixed $c \in \{0, \dots, r - 1\}$. Therefore, by the induction hypothesis, q commits to m by cycle $2(y - r + c)D - 1$. By the protocol, $q(u, v)$ sends COMMIT messages using LOCAL BROADCAST in cycle $2vD = 2(y - r + c)D$ until cycle $2(v + 1)D - 1 = (2(y - r + c) + 1)D - 1$ at the latest. By the protocol, $p(x, y)$ listens for COMMIT messages from q starting in cycle $2(y - r + c)D$ until $(2(y - r + c) + 1)D - 1$ at the latest; note that p listens to many executions of LOCAL BROADCAST containing HEARD messages, but

we focus on this particular one from q . Therefore, p and q are synchronized in the execution of LOCAL BROADCAST and p will receive q 's message by cycle $(2(y-r+c)+1)D-1 = (2(b+c+1)+1)D-1$ at the latest. Since this occurs for all nodes in A_p , node p has received all COMMIT messages from A_p by cycle $(2(y-1)+1)D-1 = (2(b+r)+1)D-1 \leq 2(b+r+1)D-1 = 2yD-1$.

Two-Hop Paths: consider the sets $B_p = \{q(u, v) \mid (a+z+1) \leq u \leq (a+r) \text{ and } (b+1) \leq v \leq (b+r)\}$ and $B'_p = \{q'(u', v') \mid (a+z+1-r) \leq u' \leq (a) \text{ and } (b+r+1) \leq v' \leq (b+2r)\}$. The set B'_p is obtained by shifting left by r units and up by r units. Recall that there is a one-to-one mapping between the nodes in B_p and the nodes in B'_p ; these are sister nodes. There are $r(r-z)$ paths of the form $q \rightarrow q' \rightarrow p$ where q and q' are sister nodes. These sets, and the notion of sister nodes, were illustrated in Chapter 1, but we depict them again here in Figure 3.4.

Consider a correct node $q(u, v) \in B_p$ and its sister node $q'(u', v') \in B'_p$ where $v' = v+r$ by definition. Again, given the location of $q(u, v)$ relative to $p(x, y)$, we have $v = y-r+c$ for some fixed $c \in \{0, \dots, r-1\}$. By the induction hypothesis, q commits to m by cycle $2vD-1$. Then by DOS-RESISTANT RELIABLE BROADCAST, q sends a COMMIT message using LOCAL BROADCAST in cycle $2vD = 2(y-r+c)D$ until cycle $2(v+1)D-1 = (2(y-r+c)+1)D-1$ at the latest. Again, this is the particular execution of LOCAL BROADCAST between q and q' ; q performs others. By DOS-RESISTANT RELIABLE BROADCAST, $q'(u', v')$ receives COMMIT messages from q using LOCAL BROADCAST starting in cycle $2(v'-r+c)D = 2vD = 2(y-r+c)D$ and ending no later than cycle $2(v'-r+c+1)D-1 = 2(v+1)D-1 = (2(y-r+c)+1)D-1$.

By the above, each node $q'(u', v') \in B'_p$ is able to start sending a HEARD message using LOCAL BROADCAST in cycle $2(v'-r)D+D$ and ending no later than cycle $2(v'-r)D+2D-1$. Starting in cycle $2(y-r+c)D+D$, node $p(x, y)$ uses LOCAL BROADCAST to listen for a HEARD message from $q'(u', v')$ where $v' = y+c$. Therefore, p is listening to q' starting in $2(y-r+c)D+D = 2(v'-r)D+D$ and ending no later than $2(v'-r)D+2D-1$; p and q' are synchronized. Therefore, p receives all HEARD messages by cycle $2(v'-r)D+2D-1$ when $v' = y+r-1$ (the node at the top row of B'_p); that is, by cycle $2(y-1)D+2D-1 = 2yD-1$.

Therefore, a total of $r(r+z+1) + r(r-z) = r(2r+1)$ node-disjoint paths from $N(a, b)$ to $PN(a, b)$ exist, all lying in in a single neighborhood $N(a, b+$

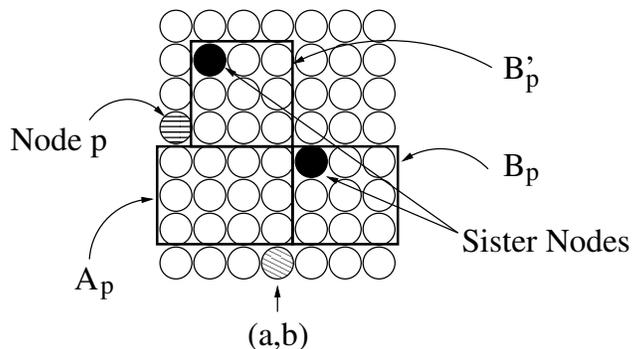


Figure 3.4: Depiction of the sets A_p , B_p , B'_p and sister nodes for a particular node p in the grid. Here $a, b, z = 0$ and $r = 3$ so the corridor C has width $2r + 1 = 7$.

$r + 1$). For an adversary corrupting $t < (r/2)(2r + 1)$ nodes, a correct node can majority filter to obtain m . Furthermore, we have shown that any $p(x, y) \in N(a, b + 1)$ executes LOCAL BROADCAST $r(2r + 1) = O(r^2) = O(t)$ times in order to receives all COMMIT and HEARD messages by cycle $2yD - 1$. Therefore, p can commit to the correct message by cycle $2yD - 1$; this concludes the induction. \square

Finally, we reiterate that proving reliable broadcast in a corridor is sufficient as the entire grid (with the exception of the boundary of width less than r in the case of a finite grid) can be covered piecewise by such corridors.

3.3.5 Reliable Broadcast in General Topologies

We examine the grid model previously because it features in previous literature on jamming-resistant reliable broadcast [4, 20, 24]. In this section, we present our results for reliable broadcast on an arbitrary graph $G = (V, E)$. Recall that Pelc and Peleg [119] examine a generalization of the t -locally bounded fault model; that is, where each node contains at most t Byzantine nodes within its neighborhood. Specifically, they examine the broadcast protocol of Koo [90], which the authors call the *Certified Propagation Algorithm* (CPA), with the aim of establishing conditions for which it achieves reliable broadcast under arbitrary graphs in contrast to the grid model. CPA does not always achieve optimal fault tolerance; for example, it cannot tolerate

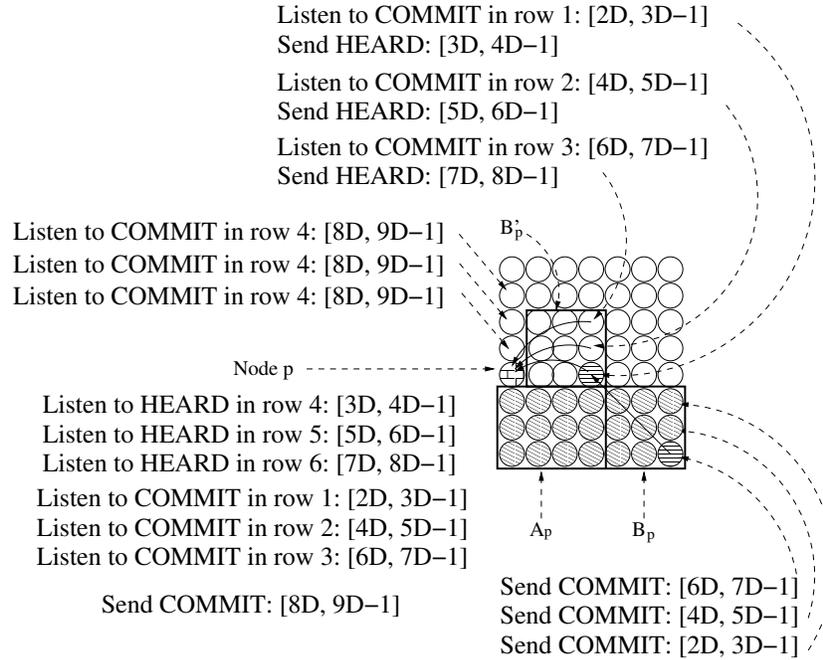


Figure 3.5: An example of some steps of the protocol for $r = 3$. The node in row 4 highlighted with the horizontal lines in the B'_p listens to a COMMIT message from its sister node in B_p by partaking in LOCAL BROADCAST as a receiver from cycle $2D$ to cycle $3D - 1$. Then, in cycle $3D$ to cycle $4D - 1$, that node uses LOCAL BROADCAST to send a HEARD message to p who is listening in this execution of LOCAL BROADCAST from cycle $3D$ to cycle $4D - 1$. The listening for p for each row is described on the left; note the synchronization. We also illustrate that those nodes above p will be listening for p 's COMMIT message using LOCAL BROADCAST at the appropriate time.

the optimal number of faults $t = (r/2)(2r + 1) - 1$ in the grid as we do above. However, we address CPA because its generality is powerful.

Again, CPA requires that all nodes obey a global broadcast schedule (i.e. there is no jamming adversary). Pelc and Peleg [119] define $X(p, d)$ to be the number of nodes in p 's neighborhood $N(p)$ that are closer to d than p and then introduce the parameter $X(G) = \min\{X(p, d) \mid p, d \in V, (p, s) \notin E\}$. To reiterate, one of their main results is that, for *any* graph G with dealer d such that $t < X(G)/2$, CPA achieves reliable broadcast. For our purposes, define for each node p the set of nodes $X(p)$ to be those $X(p, d)$ nodes closer to the dealer than p . Clearly, it is possible to identify $X(p)$ in polynomial

Certified Propagation Algorithm (Koo [90] and Pelc and Peleg [119])

- The dealer d sends the message to all of its neighbors and terminates.
- For a correct node $u \in N(d)$, upon receiving m from d it commits to m , node u announces this commitment of its neighbors and terminates.
- If a node is not a neighbor of the source, then upon receiving $t + 1$ copies of m from $t + 1$ distinct neighbors, it commits to m , and announces this commitment to its neighbors and terminates.

Figure 3.6: Pseudocode for the Certified Propagation Algorithm (CPA).

time and so we observe:

Observation 1. *If the topology of G and the location of d is known to all nodes, then each node p can calculate $X(p)$.*

A Favourable Protocol in General Topologies

The pseudocode for CPA is given in Figure 3.6. Note that, unless a node is sending in the slot allotted to it by the global broadcast schedule, or it has terminated, it is perpetually listening. We aim to remove this wasteful listening by synchronizing the sending and listening of nodes.

In the context of CPA, we call a single iteration of the global broadcast schedule a *broadcast round* (we use cycles again later on when we modify CPA). Throughout, assume that time is measured from when the dealer first broadcasts m in broadcast round 0. Under CPA, regardless of the worst-case delay imposed by the adversary, there is a broadcast round where p must have received at least $t + 1$ messages from distinct correct nodes in $X(p)$ allowing p to commit to m ; denote this broadcast round by s_p . Note, that in any execution of reliable broadcast, p may actually be able to commit before broadcast round s_p , but s_p is the maximum broadcast round in which p is guaranteed to have all the information it needs to commit to m regardless of how the adversarial nodes behave.

Since a correct node $u \in N(d)$ accepts what it hears from the dealer d immediately, and d 's broadcast round is 0, $s_u = 1$. For nodes not in $N(d)$,

the situation is slightly more complicated. In the grid, for node $p(x, y)$, we were able to compute s_p explicitly (in terms of cycles) as $2yD - 1$ in the corridor C (see proof of Lemma 11). Here, unlike with the grid, we cannot specify s_p explicitly for any graph G because it is dependent on the topology; however, by the correctness of CPA, every node eventually commits and so s_p must exist for each node p . In fact, our protocol based on CPA is simpler than that in the grid because the protocol of Bhandari and Vaidya [22] uses HEARD messages (which makes the synchronization tedious), while CPA uses only COMMIT messages.

For a *fixed* G whose topology is known to all nodes (including knowledge of where the dealer d is situated), each node p can calculate s_p . This is done by simulating the propagation of m using CPA. In this simulation, each node p has the maximum $t = X(G)/2 - 1$ Byzantine nodes in $X(p)$ and these Byzantine nodes send their faulty messages prior to the $t + 1$ correct responses in order delay propagation of m for as long as possible. By assuming that every $X(p)$ has the maximum number of Byzantine nodes, the actual placement of the Byzantine nodes in G does not affect the worst-case broadcast time s_p . In tracing this propagation, any node can calculate s_p for any node p .

Now, consider the following minor modifications to CPA: (1) each correct node p only listens to $q \in X(p)$ in broadcast round $s_q + 1$, and (2) each correct node p only sends its commit message in broadcast round $s_p + 1$. In all other slots, a node p is sleeping. This is a minor modification of CPA; call it CPA_0 . These modifications synchronize the sending/listening and allow nodes to otherwise sleep instead of perpetually listening as in CPA. The pseudocode for CPA_0 is given in Figure 3.7. While it is fairly clear that this modification does not affect correctness, we state it formally for completeness.

Lemma 12. *If CPA achieves reliable broadcast, then CPA_0 achieves reliable broadcast.*

Proof. For every node p , assume $X(p)$ has the maximum $t = X(G)/2 - 1$ Byzantine nodes and that these Byzantine nodes all send their messages to p ahead of the correct nodes in $X(p)$ according to the broadcast schedule. Pelc and Peleg [119] showed that CPA is correct in this situation (their result is independent of any particular ordering of sending in the broadcast

CPA₀

- In broadcast round 0, the dealer d sends the message to all of its neighbors and terminates.
- For a correct node $u \in N(d)$, u listens in broadcast round 0 and accepts m as correct, announces this commitment to its neighbors in broadcast round 1, and terminates.
- If a node p is not a neighbor of the source, then p listens to each neighbor $q \in X(p)$ in broadcast round $s_q + 1$; otherwise, p sleeps. Upon receiving $t + 1$ copies of m from $t + 1$ distinct neighbors in $X(p)$, it accepts m as correct, announces this commitment to its neighbors in broadcast round $s_p + 1$, and terminates.

Figure 3.7: Pseudocode for CPA₀.

schedule; that is, CPA remains correct if Byzantine nodes always send first). Therefore, in this case, each correct node p would receive a commitment from $q \in X(p)$ in broadcast round $s_q + 1$ and node p would announce its commitment in round $s_p + 1$. This is exactly what happens in CPA₀ with nodes sleeping otherwise. Therefore, if CPA achieves reliable broadcast, then so does CPA₀. \square

Define a cycle as done before in the grid. In Figure 3.8, we provide pseudocode for a fair and favourable reliable broadcast algorithm FCPA that tolerates the jamming adversary described in Theorem 3.

Lemma 13. *Assume CPA achieves reliable broadcast on a graph G . Then FCPA guarantees reliable broadcast on G .*

Proof. Using FCPA, we claim that every correct node p can commit by cycle $s_p \cdot D$. To prove this, assume the opposite: that some node p does not commit to the correct value by cycle $s_p \cdot D$. Then, there is some correct node $q \in X(p)$ that: (1) could not commit to a message by time slot $s_q \cdot D$ (and could not send p a commitment message), or (2) committed to a wrong message (and sent that wrong message to p). Note that the time for any node p' to send its commit message to a node p'' is at most D cycles by Lemma 10. Therefore, if q cannot commit (or commits to the wrong value) by $s_q \cdot D$ in FCPA,

FCPA

- The dealer d sends the message to all of its neighbors using LOCAL BROADCAST($m, d, N(d)$) and terminates after at most $D = 25 \cdot (B_0 + 2^{\varphi-1} \ln^{\varphi-1} n)^{\varphi+1}$ cycles.
- If node $u \in N(d)$, then upon receiving m from d via LOCAL BROADCAST($m, d, N(d)$), it accepts m as correct, announces this commitment to its neighbors in cycle D , and terminates.
- If a node p is not a neighbor of the source, then p listens to each neighbor $q \in X(p)$ via LOCAL BROADCAST($m, q, N(q)$) starting in cycle $s_q \cdot D + 1$ and ending by $(s_q + 1) \cdot D$; otherwise, p sleeps. Upon receiving $t+1$ copies of m in this fashion from $t+1$ distinct neighbors in $X(p)$, it accepts m as correct, announces this commitment to its neighbors using LOCAL BROADCAST($m, p, N(p)$) in broadcast cycle $s_p \cdot D + 1$, and terminates by cycle $(s_p + 1) \cdot D$.

Figure 3.8: Pseudocode for FCPA.

then q cannot commit by cycle s_q in CPA₀; therefore, CPA₀ fails to achieve reliable broadcast. However, if CPA₀ fails to achieve reliable broadcast, then by the contrapositive of Lemma 12, this contradicts the assumption that CPA achieves reliable broadcast. \square

Finally, the following analysis proves favourability, both for the grid and for general topologies:

Theorem 3 —Cost Analysis: In both of our protocols, each correct node p partakes in an execution of LOCAL BROADCAST $O(t)$ times as a sender and receiver; let k denote the total number of such executions. For the i^{th} such execution, let τ_i be the number of slots for which the adversary is active for $i = 1, \dots, k$. Denote the adversary's total active time by $\beta = \sum_{i=1}^k \tau_i \leq B_0$. Consider two cases:

Case I: Assume the adversary is active for a total of $\beta = \sum_{i=1}^k \tau_i = O(t \ln^{\varphi+1} t)$ slots over all k executions of LOCAL BROADCAST involving p . For each execution, p incurs $O(\tau_i^{\varphi-1} \ln t + \ln^{\varphi} t)$ cost in expectation by Theorem 2. Therefore, over $k = O(t)$ executions, p 's expected total cost is $O((\sum_{i=1}^k \tau_i^{\varphi-1}) \ln t + t \ln^{\varphi} t) = O((\sum_{i=1}^k \tau_i) \ln t + t \ln^{\varphi} t) = O(\beta \ln t + t \ln^{\varphi} t) = O(t \ln^{\varphi+2} t)$.

Case II: Otherwise, $\beta = \sum_{i=1}^k \tau_i = \omega(t \ln^{\varphi+1} t)$. By Jensen's inequality for concave functions, for a concave function f , $f(\frac{1}{k} \sum_{i=1}^k \tau_i) \geq \frac{1}{k} \sum_{i=1}^k f(\tau_i)$. Since $f(\tau) = \tau^{\varphi-1}$ is concave, it follows that $\sum_{i=1}^k \tau_i^{\varphi-1} \leq k(\frac{1}{k} \sum_{i=1}^k \tau_i)^{\varphi-1} = k^{2-\varphi} (\sum_{i=1}^k \tau_i)^{\varphi-1}$. Therefore, the total expected cost to p over $k = O(t)$ executions is $O((\sum_{i=1}^k \tau_i^{\varphi-1}) \ln t) + O(t \ln^{\varphi} t) = O(t^{2-\varphi} (\sum_{i=1}^k \tau_i)^{\varphi-1} \ln t) + O(t \ln^{\varphi} t) = O(t^{(2-\varphi)\beta^{\varphi-1}} \ln t + t \ln^{\varphi} t) = o(\beta)$. Therefore, p 's expected cost is less than that of the adversary.

Substituting $t = O(r^2)$ into the above analysis yields the favourability result in the grid and, together, gives our result for the grid model in Theorem 3. \square

The Las Vegas Guarantee in Multi-Hop WSNs

In addition to the rationale given in Section 3.1.1, the Las Vegas property is also valuable in multi-hop sensor networks for the following reason. Let n be the number of devices within transmitting distance of a device, and let N be the total number of devices in the network. Monte Carlo protocols that succeed with high probability in n are possible. However, typically, $n \ll N$ and messages will traverse a chain of multiple hops; consider $\Omega(N)$ hops. Consider a failure probability for a single hop that is small, but non-zero, such as $\Theta(n^{-c})$ for some constant $c > 0$, or even $\Theta(2^{-n})$, then communication fails along the chain with constant probability. Alternatively, we might achieve protocols that succeed with high probability in N . However, in large networks, N may not be known *a priori*. Furthermore, achieving a high probability guarantee in N typically involves $\Omega(\log N)$ operations which, for large N , may be too costly. Therefore, by devising Las Vegas protocols, we avoid assumptions that are problematic given that $n \ll N$.

We note that transmission over the wireless medium is subject to error due to radio-irregularity and gray-zone effects. Does this reduce the utility of our Las Vegas guarantee? In many cases, we argue that it does not. Under fair weather conditions, the percentage of successfully received packets is nearly 100% up to a distance threshold exceeding 25 meters in the case of the MICA2DOT mote [7]. Other experimental studies have shown that communication is reliable so long as the signal-to-interference-plus-noise-ratio exceeds a threshold value [143,144]; therefore, using a transmission power above this threshold yields reliable communication. Other experimental studies on

the packet reception rate, which closely approximates the probability of successfully receiving a packet between two neighbouring nodes, is perfect up to a fixed distance [173]. Therefore, for an appropriate transmission power in dense sensor networks, communication over the wireless medium should not undermine our Las Vegas guarantee.

3.4 Application 2: Application-Level DDoS Attacks

Typically in application-level DDoS attacks, a number of compromised clients, known collectively as a *botnet*, are employed to overwhelm a server with requests. These botnets have become commercialized with operators (“bot-masters”) renting out time to individuals for the purposes of launching attacks [54, 96].

We assume a model of botnet attacks similar to that described by Wal-fish *et al.* [152]. In this model, a request is cheap for a client to issue, expensive for the server to service, and all requests incur the same computational cost (heterogeneous requests can likely be handled as in [152]). There is a high-capacity communication channel and the crucial bottleneck is the server’s inability to process a heavy request load.

The client rate is g requests per second. The aggregate botnet rate is R requests per second and this is assumed to be both relatively constant and the botnet’s maximum possible rate. If the server is overloaded, it randomly drops excess requests. In this case, the good clients only receive a fraction $g/(g+R)$ of the server’s resources; it is assumed that $R \gg g$ so that $g/(g+R)$ is very small.

Wal-fish *et al.* [152] propose a protocol SPEAK-UP for resisting DDoS attacks by having clients increase their sending rate such that their aggregate bandwidth G is on the same order as that of R . Since botnet machines are assumed to have already “maxed-out” their available bandwidth in attacking, SPEAK-UP greatly increases the chance that the server processes a legitimate request since $G/(G+R) \gg g/(g+R)$. A crucial component of SPEAK-UP is a front-end to the server called the “thinner” which controls which requests are seen by the server and asks a client to retry her request if it was previously dropped.

<p>DOS-RESISTANT CLIENT-SERVER COMMUNICATION for round $i \geq 2$</p> <p><i>Send Phase:</i> For each of the 2^{2i} slots do</p> <ul style="list-style-type: none"> • The client sends her request with probability $2/2^i$. • The server (via the thinner) admits listens with probability $2/2^i$. <p><i>Ack Phase:</i></p> <ul style="list-style-type: none"> • The server sends back the requested data. • The client listens. <p>If the client receives her data, she terminates; otherwise, the thinner tells her to retry in the next round.</p>
--

Figure 3.9: Pseudocode for the application of Case 2 of our 3-Player Scenario to the client-server scenario.

3.4.1 Our Protocol

We employ Case 2 of our 3-PLAYER SCENARIO PROTOCOL to achieve a SPEAK-UP-*like* algorithm with provable guarantees. Bandwidth (upstream and downstream rates in bits per second) is our measure of cost and, as such, our results should be interpreted as quantifying the expected upstream bandwidth required by the client and the expected downstream bandwidth with which the server should be provisioned. Using bandwidth as a form of currency has been previously employed by the research community [69, 140, 152]. Our pseudocode is given in Figure 3.9.

Note that the Ack Phase is simplified due to the fact that attacks do not occur in this phase for Case 2 of the 3-PLAYER SCENARIO PROTOCOL. Like [152], our protocol is suitable for applications where there is no pre-defined clientele (so the server cannot traffic filter) and the clientele can be non-human (so “proof-of-humanity” tests cannot be relied upon solely). Unlike the wireless domain, we do not address reactive adversaries. Determining when a player is sending over the wire in order to control when its traffic arrives at the targeted player seems beyond the capability of a realistic attacker.

The client plays the role of Alice where the message is a request; the server plays the role of Bob. This application falls into Case 2 of Theorem 1: a DDoS attack targets the server while communications from the server to the clients are not disrupted. The client and server are assumed to be synchronized such that they always agree on the current round and a maximum

round number is set *a priori*. Such synchronization is certainly possible over Internet-connected machines and the maximum round value should be set to account for the level of DDoS resistance the participants wish to have; for most attacks, R is in the low hundreds of Mbits/second [139]. We now provide an overview of our protocol.

Send Phase: Each Send Phase occurs over a uniform and fixed duration Δ ; for simplicity, we set $\Delta = 1$ second, and the slot length changes in each round appropriately. The client sends in each slot with probability $2/2^i$ with an expected 2^{i+1} upstream bits per second. The server listens in each slot with probability $2/2^i$ for an expected 2^{i+1} downstream bits per second. If the received traffic substantially exceeds 2^{i+1} , requests are dropped; probabilistic listening and traffic measurement on the server side can be performed by the thinner [152].

Note that in each round, the client increases her sending rate in the Send Phase to “speak up”. Any correct client that reaches its bandwidth limit remains at this limit for the duration of the protocol. When the maximum round number is reached, the clients maintain their sending rate until the thinner informs them that the attack has ended. For the purposes of analysis, a blocked slot occurs when Carol overwhelms the server with requests and the client’s request is dropped in that slot. Define a send-blocked phase as one where Carol blocks at least $2^{2i}/2$ slots; therefore, Carol uses an upstream bandwidth of *at least* $2^{2i}/2$ bits per second. As in [152], if the thinner drops a request, it immediately asks the client to retry in the next round.

Ack Phase: The server does not increase its sending rate per round (only the client speaks up) since there are no attacks in the Ack Phase for Case 2. This simplifies the Ack Phase as mentioned in Section 3.2 in our discussion of Ack Failures; the server simply returns the requested data to the client at some reasonable rate.

We assume upstream and downstream bandwidth are capped; this is true of residential Internet packages, as well as hosted services. In the case of residential service, upstream bandwidth is scarcer than downstream bandwidth, while servers are generally well-provisioned for both; this can be reflected in our cost constants. By Case 2 of Theorem 1 we have:

Corollary 1. *If Carol uses bandwidth R to attack, then the client’s request is serviced, and the expected bandwidth (upstream and downstream) used by the client and the server is $O(R^{0.5})$.*

Bob can represent multiple good clients. We assume the same synchronization with the server; however, clients joining at different times are informed by the thinner of the current round. In order to be guaranteed *some* of the server’s resources, the clients’ expected aggregate bandwidth is $G = \Omega(R^{0.5})$. Therefore, our result quantifies the minimum expected aggregate upstream bandwidth for clients and the expected downstream bandwidth for the server required to ensure that total censorship is averted; in contrast, SPEAK-UP cannot make such a guarantee. This is useful for applications where a critical update or warning must be disseminated, and delivery to even a handful of clients is sufficient since they may then share it with others (via multicast, peer-to-peer distribution, etc.).

As with SPEAK-UP, the probability of legitimate request being serviced is still $G/(G + R)$. In addition to admitting an analysis, our iterative approach of geometrically increasing the aggregate bandwidth should mitigate attempts by Carol at launching short duration DDoS attacks in order to provoke a steep and disruptive traffic increase from correct clients. Our protocol is fair as described in Section 3.1.2—the *aggregate* requirements of the bandwidth constrained clients is asymptotically equal to that of the well-provisioned server. Restating our result above in the context of multiple clients yields Theorem 4.

Finally, in order to achieve the same level of denial of service against a server that is defended by our protocol, Carol must procure a much larger botnet in order to obtain the necessary bandwidth; however, this comes at a cost. For example, one study found the cost of a single bot to be between \$2 and \$25 [54]. Therefore, since Carol’s bandwidth requirements increase quadratically, her monetary costs increase significantly with the use of our protocol.

Chapter 4

Reducing Listening Costs for Reliable Broadcast in Wireless Sensor Networks

Energy is one of the most critical resources in wireless sensor networks. The wireless radios on sensor network devices offer a number of different modes typically with states such as *off*, *sleeping*, *idle*, *receiving* and *sending* [158]. Remarkably, the cost of the idle, receiving, and sending states are roughly equivalent, and these costs are an order of magnitude larger than the cost of the sleep state. The difference in energy consumption between the idle/send/receive states and the sleep state differs depending on the type of card and the communication standard being employed. For example, using the IEEE 802.11 standard with a 11 Mbps card, the ratios between energy consumption of the idle/send/receive states and the sleep state are all more than 12 [50]. In [76], with a different setup employing TinyOS and a TR1000 transceiver, the measured ratios are over 1000. Therefore, the amount of time spent in the sleep state is strongly indicative of the energy efficiency of a given algorithm [156]. Here, we consider a node to be either asleep or awake (listening and/or sending). Our goal is to design an algorithm that allows a single node to broadcast a message so that eventually all non-faulty nodes learn the correct message; this is the problem of reliable broadcast. Almost all previous work on the reliable broadcast problem ignores energy efficiency, assuming the nodes are spending a substantial amount of time listening. Here, we do not address the issue of relative energy costs as we did

in Chapter 3. Instead, we focus on designing algorithms for reliable broadcast that reduce the absolute energy costs to correct nodes, particularly with regards to the cost of listening. Our approach depends upon the analysis of a new data streaming problem that we call the Bad Santa problem.

4.1 The Bad Santa Problem

We restate the Bad Santa problem here along with another variant that we consider in this chapter. A child is presented with n boxes, one after another. When given each box, the child must immediately decide whether or not to open it. If the child decides not to open a box, he is never allowed to revisit it. At least half the boxes have presents in them, but the decision as to which boxes have presents is made by an adversarial Santa who wants the child to open as many empty boxes as possible. The child wants to find a present, while opening the smallest expected number of boxes.

More formally, an adversary sends a stream of n bits of which at least half are 1. The adversary sets the bits of the stream prior to sending the first bit. The algorithm may query any bit as it passes, but once a bit passes without being queried, it is lost. The algorithm is correct if it always finds a 1. The adversary knows the (randomized) algorithm ahead of time but not its random bits. The cost of an algorithm on an input is the number of expected queries executed until it finds a 1. The goal is to design a correct algorithm with minimum expected cost over the worst case input. At first glance, it may appear that randomly sampling $O(\log n)$ presents trivially solves the single stream Bad Santa problem. However, this strategy has a (small) probability of failure, which is unacceptable and we elaborate on this later in Section 4.2.1.

We are interested in two variants of this problem. The first is the single stream case described above. The second is the multi-stream case where there are multiple n -bit streams that the algorithm queries consecutively. Each stream has a constant fraction of 1 bits, but the values (1s and 0s) may be distributed differently in each stream; note that, in the multi-stream case, the fraction of 1 bits can be less than $1/2$. A correct algorithm must find one 1 bit in one of the streams. The cost is the expected number of queries over the worst case set of such streams.

4.1.1 The Grid Model of Sensor Networks

We demonstrate the applicability of the Bad Santa problem on the grid network model. We make some additional remarks regarding the flexibility of the grid model later on in Section 4.4.8.

Faults

Every node in the grid may suffer faults, but as in [21, 22, 24, 90] we assume that no more than t nodes in any $2r + 1$ by $2r + 1$ square are faulty and that no node can spoof another node's identity. We consider the cases where these faults are either all *fail-stop*: the t nodes are all deleted from the network; or *Byzantine*: the t nodes are taken over by an adversary and deviate from our protocol arbitrarily; although, Byzantine nodes must also abide by the schedule as in [21, 22, 90]. Note that tolerating a known bounded number of deviations from the schedule is possible (see [24]) and that our results can likely be made resilient in the same fashion. We assume that all of the nodes that suffer faults are chosen by a single adversary who controls these nodes to coordinate attacks on the network. This adversary knows everything except for the random bits of the non-faulty nodes.

Schedule of Transmissions

We assume there is a distinguished node s known as the *source* (we use this term rather than *dealer* throughout) that holds an initial message m . We assume without loss of generality that the source node has coordinates $(0, 0)$ on the grid, i.e. *all nodes know the source*. We discuss relaxing this assumption in Section 4.2.3. All known protocols designed for the reliable broadcast grid model proceed in steps where the source of the message sends to its neighbors, which in turn send to their neighbors, until all nodes receive the message. The predecessor set G_p of a correct node p is a particular set of nodes such that if p listens to all nodes in G_p and majority filters on the received messages, p will obtain the correct message; we give a precise definition of G_p in Section 4.4.5. Again, following the literature, we assume that each node has a predecessor set of $n = r(2r+1)$ nodes assigned to distinct time slots and that the entire schedule repeats every $(2r + 1)^2$ time slots. We call each schedule repetition a *round*. An example of a broadcast schedule

is given in [90]: In each round, each node in position (x, y) broadcasts in time slot $((x \bmod (2r + 1)) \times (2r + 1) + (y \bmod (2r + 1))) \bmod (2r + 1)^2$. For our purposes, it suffices to assume each round has $O(n)$ time slots and each node within L_∞ distance r of some node p is assigned to a distinct time slot. If $t < n/2$ then each node has a predecessor set of which strictly less than half of the nodes are faulty, or it can listen directly to the source which we assume is correct [21, 22]. For simplicity, we assume the source initially broadcasts the message size and, thereafter, time slots are long enough to send the entire message.¹ The cost of listening to a message is proportional to the message length.

4.2 The Bad Santa Problem and Reliable Broadcast

The Bad Santa problem is an abstraction of the second single-hop communication scenario described in Chapter 1 where a set of multiple senders possess information m required by a single receiver. However, many of the senders (strictly less than half) can be faulty, failing to send m or sending incorrect data. While the majority of the senders are correct, rather than listening to all of \mathcal{S} and majority filtering on the received data, we desire an algorithm that allows the single receiver to decide on m in a more efficient manner. We now sketch the methods for applying the solutions of the Bad Santa problem to the problem of reliable broadcast.

Our goal is to reduce the expected listening time and the expected bit complexity required for a node to learn the message from its n predecessors. We can use the algorithm for the single stream Bad Santa problem to do so *provided* that: (1) at least half of the predecessors have the correct message and, in the case of Byzantine faults, the listening node can determine if a message is correct; (2) the listening node knows the location of the source node and time of broadcast (to determine when to start the Bad Santa protocol and to which set of n nodes to possibly listen). In this case, the message

¹An alternative is that the source node preprocesses the message by dividing it into pieces that each fit into a time slot. However, both the broadcasting of the message size and the details of how the message might be formatted for sending are outside the scope of this work.

can be transmitted safely from one set of predecessors to another, with each node using the Bad Santa protocol to decide which of its predecessors to listen to and thereby learn the message.

We can reduce listening time further by using the multi-stream Bad Santa protocol. Here the fraction of faulty predecessors can exceed $1/2$ and we show that multiple streams are required if we wish to obtain savings. If we use $k + 1$ streams, then there are $k + 1$ rounds of sending before the message is passed from one set of nodes to another; each node sends $k + 1$ times and the latency increases by a factor of $k + 1$ over the single round case.

4.2.1 Utility of Las Vegas Algorithms

Why do we insist on allowing no error in the Bad Santa problem? Why not just use random sampling? We reiterate our argument from Section 3.1.1. Random sampling has a probability of error that depends on n , which is on the order of the number of nodes in the transmission radius; we stress that n depends on r and is *not* the total number of nodes in the network. If the network's total size is much larger than n , then even if the failure probability for a single listener is exponentially small in n , the probability that some node in the network fails to learn the message will still be quite large. For example, if the total network size is exponential in n and the probability of failure for a single listener is small, but non-zero, say $\Theta(2^{-n})$, then with constant probability, reliable broadcast will fail.

4.2.2 Byzantine Faults: Known Start Time and Source

To satisfy condition (1) when the faults are Byzantine, our protocol has two stages. In the first stage, the source uses a secure (cryptographic) hash function (for more on such hash functions see [147], Chapter 4) to generate a fingerprint of size $(\log_2 |m|)^2$ where $|m|$ is the message length², and broadcasts this fingerprint to all the other nodes in the network using a previously known energy-inefficient method in [22]. In the second stage, the source broadcasts the full message with each node using a Bad Santa protocol. Each node compares the hash value of each full message received against the

²We make the random oracle assumption about the hash function used to generate the fingerprint of m .

true fingerprint to determine if it agrees and is thus presumably correct. If the adversary is unable to discover a false message whose hash matches the fingerprint, then the only message which matches the fingerprint is the correct message. Each node can determine if the message it receives is correct. Thus, at each stage, all non-faulty nodes transmit the correct message and condition (1) is satisfied. This introduces a possibility of error into the transmission which depends on the relative size of the fingerprint to the message and the resources of the adversary. In this model, the set of faulty nodes can differ from one stream to the next as chosen by the adversary; however, for a given stream, the adversary must decide whether a node is corrupt prior to its selection or non-selection by a protocol.

4.2.3 Byzantine Faults: Unknown Start Time and Source(s)

We also deal with the case where the start time of the message is not known in advance, or the location of the source is not known. Moreover, our protocol allows any node to send a message i.e. become a source node. We note that this is also possible under the original protocols of [21, 22, 90]; however, we explicitly deal with this case and show how to accomplish an energy savings if $t < \frac{n}{16+\epsilon}$ for any constant $\epsilon > 0$. More specifically, we require that no more than a $1/2 - \epsilon$ fraction of the nodes are faulty in any $r/2$ by $r/2$ square. In this model, the adversary is adaptive in the sense that it can decide which nodes to take over based on which nodes have previously committed to the correct message.

4.3 Our Contributions

Our five main results are summarized in the theorems below. For ease of exposition, we have aggregated the notation we most commonly use throughout this chapter in Table 4.1. Finally, throughout, let $\lg n$ denote the logarithm base 2 and let $\log^{(k)} n$ denote $\underbrace{\log \cdots \log}_k n$.

Theorem 6. *For the single-stream Bad Santa problem, the optimal expected number of queries is $\Theta(\sqrt{n})$.*

Notation	Definition
r	Radius of broadcast for all nodes.
t	Number of Byzantine peers in a $(2r + 1) \times (2r + 1)$ square of the sensor network.
$p(x, y)$	A node p located at coordinate (x, y) in the grid network model.
$N(p)$ or $N(x, y)$	Set of nodes within the broadcast radius of node $p(x, y)$.
n	In the context of the Bad Santa Problem, n is the number of boxes in a stream. In the context of a sensor network, n is the size of predecessor set where $n = r(2r + 1)$.
k	Number of streams used in the problem definition of the Bad Santa Problem.
s	Source node (or dealer) in the problem of reliable broadcast.
m	Message sent by the source node in the problem of reliable broadcast.
$ m $	Number of bits in the message m .
f	A secure hash function.
$f(m)$	Fingerprint resulting from applying the hash function f to m .

Table 4.1: Summary of frequently used notation in Chapter 4.

Theorem 7. *For the k -stream Bad Santa problem, the optimal expected number of queries is $O(\log^{(k)}(n) + k)$ and $\Omega(\log^{(2k)} n)$. In particular, for $k = \Theta(\log^* n)$, we can ensure the expected number of queries is $O(\log^* n)$.*

The next two theorems about energy-efficient broadcast are established by algorithms based on solutions to the Bad Santa problem. We again repeat that $n = r(2r + 1)$ and so n depends on the broadcast radius; it is not the total number of nodes in the network. The algorithms apply to a grid of finite or infinite size. In the former case, we achieve the standard result that all nodes, except those on the boundary of width r , commit to the correct message. In the latter case, for Byzantine faults, our result translates into a finite portion of the grid obtaining the correct message and this is dependent on the computational power of the adversary. Theorem 8 essentially follows directly from Theorems 6 and 7. Theorem 9 requires a fingerprint of the message to first be broadcast through the network.

Theorem 8. *Assume we have a network where at most $t < \frac{r}{2}(2r + 1)$ nodes*

suffer fail-stop faults in any square of size $2r + 1$ by $2r + 1$ and that the start time and source of a message are known. Then there exists an algorithm for reliable broadcast which has the following properties:

- Each node is awake for $O(\sqrt{n})$ time slots in expectation.
- Each node broadcasts $O(\sqrt{n}|m|)$ bits and receives $|m|$ bits.

In the next theorem, we use the notion of *computational steps* in the context of the adversary. By this, we mean the number of times the adversary can create an input x' , apply a secure hash function f to x' and check for a match between the output fingerprint $f(x')$ and some other fingerprint for which the adversary is attempting to generate a collision.

Theorem 9. *Assume we have a network where at most $t < \frac{r}{2}(2r + 1)$ of the nodes suffer Byzantine faults in any square of size $2r + 1$ by $2r + 1$ and that the start time and source of a message are known. Further assume that the number of computational steps available to the adversary is bounded by s . Then there exists an algorithm for guaranteeing reliable broadcast with a probability of failure $O(s/|m|^{\lg |m|})$. In an initial stage, the algorithm requires a fingerprint of size $\lg^2 |m|$ to be initially broadcast to the network. However, in the second stage, when the message m itself is broadcast, the algorithm has the following properties:*

- Each node is awake for $O(\sqrt{n})$ time slots in expectation,

Over both stages, the algorithm has the following costs:

- Each node broadcasts $O(n \log^2 |m| + \sqrt{n}|m|)$ bits and receives an expected $O(n \log^2 |m| + \sqrt{n}|m|)$ bits.

We also present results on increased energy savings for values of t within an arbitrary constant factor of optimal. In particular, we consider the case where $t \leq (1 - \epsilon)\frac{r}{2}(2r + 1)$ for any constant $\epsilon > 0$ where we have the following results:

Theorem 10. *Assume we have a network where, for any constant $\epsilon > 0$, at most $t \leq (1 - \epsilon)\frac{r}{2}(2r + 1)$ nodes suffer fail-stop faults in any square of size $2r + 1$ by $2r + 1$ and that the start time and source of a message are known. Then there exists an algorithm which guarantees reliable broadcast and which has the following properties:*

- *For any k between 1 and $\ln^* n$, the algorithm requires each node to be awake for an expected $O(\log^{(k)} n)$ time slots.*
- *Each node broadcasts $O(k|m|)$ bits and receives $|m|$ bits.*

Therefore, the above algorithm requires each node to broadcast $O(k)$ times which translates into a higher latency given that nodes must adhere to a broadcast schedule; however, nodes save more energy in expectation.

Theorem 11. *Assume we have a network where, for any constant $\epsilon > 0$, at most $t \leq (1 - \epsilon)\frac{r}{2}(2r + 1)$ of the nodes suffer Byzantine faults in any square of size $2r + 1$ by $2r + 1$ and that the start time and source of a message are known. Further assume that the number of computational steps available to the adversary is bounded by s . Then there exists an algorithm which guarantees reliable broadcast with a probability of failure $O(s/|m|^{\lg |m|})$. In an initial stage, the algorithm requires all nodes to be awake for every slot during which a fingerprint of size $\lg^2 |m|$ is initially broadcast to the network. However, in the second stage, when the message m itself is broadcast, the algorithm has the following properties:*

- *For any k between 1 and $\ln^* n$, requires all nodes to be awake an expected $O(\log^{(k)} n)$ time slots.*

Over both stages, the algorithm has the following costs:

- *For any k between 1 and $\ln^* n$, each node broadcasts $O(n \log^2 |m| + k|m|)$ bits and receives an expected $O(n \log^2 |m| + (\log^{(k)} n)|m|)$ bits.*

Finally, we deal with the case where the start time and the source of the message is unknown. In this situation, if $t < \frac{n}{16+\epsilon}$, we have the following result:

Theorem 12. *If the start time and source of a message are unknown, there is a protocol for reliable broadcast in which each node (1) sends $O(|m|)$ bits per round, (2) is awake an amortized constant number of time slots per round and (3) receives an amortized $O(|m|)$ bits per round.*

For this last result given in Theorem 12, all nodes may receive the message; that is, those nodes on the boundary are not excluded as with our previous results.

To contrast our results with previous work, we note that under the previous algorithms for reliable broadcast [21, 22], each node 1) is awake for $(2t + 1) = \Theta(n)$ time slots, 2) broadcasts $\Theta(|m|)$ bits; 3) receives $\Theta(|m|)$ bits in the fail-stop model; and 4) can be forced by the adversary to receive $\Theta(n|m|)$ bits in the Byzantine fault model. Therefore, in both fault models, our algorithms are saving substantially on the amount of time a node must be awake for listening to the *full message*. For the fail-stop case, we are trading a small factor increase in traffic for these savings. Moreover, in the Byzantine case, we greatly reduce the total bit complexity.

Finally, note that $|m|$ need not be large to make the probability of finding a message with the same fingerprint very small. For example, if $|m| = 1$ KB, the probability of a collision is already $2^{-100} \approx 10^{-33}$.

4.4 Related Work

The reliable broadcast problem over the sensor network model described above has been extensively studied in [21–24, 90]. In [90], Koo showed that reliable broadcast with Byzantine faults is impossible if $t \geq \frac{r}{2}(2r + 1)$ in the L_∞ norm. In [21, 22], Bhandari and Vaidya presented a clever algorithm that achieved reliable broadcast tolerating Byzantine faults for any $t < \frac{r}{2}(2r + 1)$; our Theorem 9 applies to this scenario. There the authors also achieve $t < r(2r + 1)$ for the fail-stop fault model whereas our result applies only when $t < \frac{r}{2}(2r + 1)$ or when $t \leq (1 - \epsilon)(r/2)(2r + 1)$ for any constant $\epsilon > 0$. Therefore, we are a constant factor from the optimal tolerance in the fail-stop model. Koo et al., in [24], described an algorithm that achieves reliable broadcast even when the faulty nodes can spoof addresses of honest nodes or cause collisions; this is a more challenging fault model than is addressed in our work or in any other previous work. All prior algorithms

proposed for the reliable broadcast problem require each node in the network to be awake for a constant fraction of the time slots and thus are not energy efficient. Our algorithm from Theorem 9 makes use of the algorithm from [22] to broadcast a fingerprint of the message. Finally, under different models of a sensor network, the problems of consensus [35, 62], reliable broadcast under the fail-stop fault model [91] and reliable broadcast under adversarial faults [119] have been studied. Work in [57] deals with broadcast protocols in a time-slotted network where the number of times a node can transmit is constrained; this is called “k-shot broadcasting”. The authors focus on establishing bounds on the number of rounds each node must transmit in order to achieve broadcast; hence, there is a focus on the tradeoff between energy (i.e. the number of shots needed) and latency of the broadcast. However, despite this similarity, the network model used in [57] does not incorporate adversarial behaviour and captures more general topologies; consequently, the techniques and results differ significantly from our work.

Data streaming problems have been popular in the last several years [73, 110]. Generally, past work in this area focuses on computing statistics on the data using a small number of passes over the data stream. In [73], the authors treat their data stream as a directed multi-graph and examine the space requirements of computing certain properties regarding node degree and connectedness. Munro and Paterson [109] consider the problem of selection and sorting with a limited number of passes over one-way read-only memory. Guha and McGregor [66, 67] examine the problem of computing statistics over data streams where the data objects are ordered either randomly or arbitrarily. Alon, Matias and Szegedy [5] examine the space complexity of approximating the frequency of moments with a single pass over a data stream. In all of these cases, and others [37, 43], the models differ substantially from our proposed data streaming problem. Rather than computing statistics or selection problems, we are concerned with the guaranteed discovery of a particular value, and under our model, expected query complexity takes priority over space complexity.

4.4.1 Single-Stream Variant of the Bad Santa Problem

We now consider the single-stream Bad Santa problem. A naive algorithm is to query $n/2 + 1$ bits uniformly at random. The expected cost for this algorithm is $\Theta(n)$ since the adversary will place the 1’s at the end of the

stream. The following is an improved algorithm.

SINGLE STREAM SELECTION STRATEGY

- 1: Perform \sqrt{n} queries uniformly at random from the first half of the stream. Stop immediately upon finding a 1.
- 2: If no 1 has been found, starting with the first bit in the second half of the stream, query each consecutive bit until a 1 is obtained.

Lemma 14. *The expected cost of the above strategy is $O(\sqrt{n})$.*

Proof. Assume that there are $i\sqrt{n}$ 1s in the first half of the stream where $i \in [0, \frac{\sqrt{n}}{2}]$. This implies that there are then $(n/2) - i\sqrt{n}$ 1s in the second half of the stream. By querying \sqrt{n} slots uniformly at random in the first half of the stream, the probability that the algorithm fails to obtain a 1 in the first half is no more than:

$$\left(1 - \frac{i\sqrt{n}}{(n/2)}\right)^{\sqrt{n}} = \left(1 - \frac{2i}{\sqrt{n}}\right)^{\sqrt{n}}$$

for an expected overall cost not exceeding:

$$\sqrt{n} + \left(1 - \frac{2i}{\sqrt{n}}\right)^{\sqrt{n}} \cdot i\sqrt{n}.$$

We find the maximum by taking the derivative:

$$\frac{d}{di} \left(1 - \frac{2i}{\sqrt{n}}\right)^{\sqrt{n}} \cdot i\sqrt{n} = \sqrt{n} \left(1 - \frac{2i}{\sqrt{n}}\right)^{\sqrt{n}} - 2i\sqrt{n} \left(1 - \frac{2i}{\sqrt{n}}\right)^{\sqrt{n}-1}$$

and setting it to zero while solving for i gives $i = \frac{\sqrt{n}}{2(\sqrt{n}+1)}$. Plugging this into the expected cost function gives an expected cost of $O(\sqrt{n})$. \square

The following lemma was shown in [87] and, for completeness, the proof is included in the appendix of this thesis.

Lemma 15. $\Omega(\sqrt{n})$ *expected queries are necessary in the single stream case.*

Theorem 1 follows immediately.

4.4.2 Multi-Stream Variant of the Bad Santa Problem

We define a (α, β) -strategy to be an algorithm which occurs over no more than α streams, each with at least a (possibly different) set of at least $\Theta(n)$ values of 1, and which incurs expected cost (number of queries) at most β . To be explicit, for multiple streams, we can handle the case where the fraction of boxes that contain a 1, denoted by δ , can be less than $1/2$. The previous section demonstrated a $(1, O(\sqrt{n}))$ -strategy. We now consider the following protocol over $(k + 1)$ streams for $k \geq 1$.

MULTI-STREAM SELECTION STRATEGY

- 1: **for** $(i = k \text{ to } 1)$ **do**
- 2: Perform $\frac{1}{\delta} \ln^{(i)}(n)$ queries uniformly at random over the entire stream. Stop if a 1 is obtained.
- 3: If no value of 1 has been found, then open each of the n boxes in order in the final stream until a 1 is located.

Lemma 16. *For any constant $\delta > 0$, the above protocol is a $(k+1, O(\log^{(k)}(n) + k))$ -strategy.*

Proof. Correctness is clear because in the worst case, we open all boxes in the final stream. The expected cost is:

$$\begin{aligned}
 &\leq \delta^{-1} \ln^{(k)} n + \left[\sum_{i=k-1}^1 (1 - \delta)^{\delta^{-1} \ln^{(i+1)} n} \cdot O\left(\delta^{-1} \ln^{(i)} n\right) \right] + (1 - \delta)^{\frac{\ln n}{\delta}} \cdot O(n) \\
 &\leq \delta^{-1} \ln^{(k)} n + \left[\sum_{i=k-1}^1 e^{-\ln^{(k)} n} \cdot O\left(\delta^{-1} \ln^{(k-1)} n\right) \right] + e^{-\ln n} \cdot O(n) \\
 &= O(\log^{(k)}(n) + k)
 \end{aligned}$$

□

Lemma 17. *If there are $\ln^*(n) + 1$ streams and δ is a constant, then the multi-stream algorithm provides a $(O(\log^* n), O(\log^* n))$ -strategy.*

Proof. By the definition of the iterated logarithm:

$$\ln^* n = \begin{cases} 0 & \text{for } n \leq 1 \\ 1 + \ln^*(\ln n) & \text{for } n > 1 \end{cases}$$

if $k = \ln^* n$, we can plug this value into the last line of the proof of Lemma 16 which contains two terms inside the big-O notation. By the definition of $\ln^* n$, the first term is $O(\ln^{(\ln^* n)} n) = O(1)$, and the second is $O(\ln^* n)$, for a total expected cost of $O(\ln^* n)$. \square

4.4.3 Lower Bound for Multiple Streams

The following lemma is proved in [87] and is included in the appendix for completeness:

Lemma 18. $\Omega(\log^{(i+2)} n)$ expected queries are required for a randomized algorithm that errs with probability less than $\lambda = (\ln^{(i)} n)^{-\epsilon}$ on one stream of length n . In particular, when $i = 0$, $\Omega(\log \log n)$ expected queries are required for a randomized algorithm with error less than $1/n^\epsilon$, for any constant $\epsilon > 0$.

Using Lemma 18, we now achieve the following lower bound:

Lemma 19. For $k > 0$, $\Omega(\ln^{(2k)} n)$ expected queries are necessary to find a 1 from $k + 1$ streams with probability 1.

Proof. We use induction on the number of streams:

Base Case: Let $k = 1$. Either the algorithm finds a 1 in the first pass or the second pass. From Lemma 18, for any constant ϵ any algorithm that fails to find a 1 in the first pass with probability $\leq n^{-\epsilon}$ has expected cost $\Omega(\log \log n)$. If the algorithm fails to find a 1 in the first pass with probability at least $n^{-\epsilon}$ then the expected cost to the algorithm is at least the probability it fails in the first pass times the expected cost of always finding a 1 in the second and final pass, which is $n^{-\epsilon} \cdot \Omega(\sqrt{n})$. Choosing $\epsilon < 1/2$, the expected cost is $\Omega(\log \log n)$.

Inductive Hypothesis: For $k > 1$, $\Omega(\ln^{(2k)} n)$ expected queries are necessary to find a 1 from $k + 1$ streams with probability 1.

Inductive Step: Now assume the hypothesis is true for up to $k > 1$ streams. Assume we have $k + 1$ streams. Any randomized algorithm either fails to find a 1 in the first stream with probability less than $(1/\ln^{(2k-2)} n)^\epsilon$, in which case by Lemma 5, the expected cost of the algorithm when it processes the first stream is $\Omega(\ln^{(2k)} n)$ or the probability that it fails in the first pass is at least $(1/\ln^{(2k-2)} n)^\epsilon$. In that case, the expected cost deriving from queries of the second stream is at least $(1/\ln^{(2k-2)} n)^\epsilon \cdot \Omega(\ln^{(2k-2)} n)$ where the second factor of this expression is the expected number of queries needed to find a 1 in k streams, as given by the induction hypothesis. The minimum expected cost of any randomized algorithm is the minimum of these two possibilities, which is $\Omega(\ln^{(2k)} n)$. \square

Theorem 7 then follows immediately from Lemmas 3, 4, 5 and 6.

4.4.4 An Extension to Adaptive Adversaries

In this section, we take a slight detour and consider a variation of the Bad Santa problem where the adversary does not necessarily need to make all of its decisions ahead of time; we call this an adaptive adversary. Prior to the child's current selection, the Bad Santa may reorder the boxes based on the previous actions of the child. Consider the following protocol for the single stream variant of the Bad Santa Problem with an adaptive adversary:

SINGLE STREAM ADAPTIVE BAD SANTA ALGORITHM

- 1: **for** ($i = 1$ to $n/2$) **do**
- 2: Open box i with probability $2/\sqrt{n}$. Stop immediately if a toy is obtained.
- 3: Otherwise, open each box consecutively until a toy is obtained.

This algorithm differs from that in [87] in that we no longer insist on opening precisely \sqrt{n} boxes in the first half of the stream. Instead, each box in the first half of the stream is opened independently with probability $2/\sqrt{n}$.

Lemma 20. *In the presence of an adaptive adversary, the SINGLE STREAM ADAPTIVE BAD SANTA ALGORITHM FOR $\delta = 1/2$ guarantees the child obtains a toy with $O(\sqrt{n})$ cost in expectation.*

Proof. Let X be the number of boxes the child opens in the first half of the stream. For $j = 1, \dots, n/2$, let the variable $q_j = 1$ if the j^{th} box contains a toy; otherwise, let $q_j = 0$. The expected cost is then:

$$\begin{aligned} &\leq \sqrt{n} + \prod_{j=1}^{n/2} \left(1 - \frac{2}{\sqrt{n}} q_j\right) \sum_{j=1}^{n/2} q_j \\ &\leq \sqrt{n} + e^{-\frac{2}{\sqrt{n}} \sum_{j=1}^{n/2} q_j} \cdot \sum_{j=1}^{n/2} q_j \end{aligned}$$

The last factor $\sum_{j=1}^{n/2} q_j$ is the number of empty boxes in the second half of the stream (there are $n/2 - \sum_{j=1}^{n/2} q_j$ full boxes in the second half of the stream; therefore, there are $n/2 - (n/2 - \sum_{j=1}^{n/2} q_j)$ empty boxes in the second half). Let $q = \sum_{j=1}^{n/2} q_j$. Setting the derivative (with respect to q) to zero allows us to find the maximum at $\sqrt{n}/2$. Plugging this back into our equation for expected cost yields $O(\sqrt{n})$. \square

We can also handle $k + 1$ streams:

$k + 1$ STREAMS ADAPTIVE BAD SANTA ALGORITHM

- 1: **for** stream ($i = k$ to 1) **do**
- 2: **for** ($j = 1$ to n) **do**
- 3: Open box j with probability $\frac{\ln^{(i)} n}{\delta n}$. Stop immediately if a toy is obtained.
- 4: If no toy has been obtained, open all boxes in the final stream until a toy is obtained.

Lemma 21. *In the presence of an adaptive adversary, the $k + 1$ STREAMS ADAPTIVE BAD SANTA ALGORITHM for any constant $\delta > 0$ and $k \geq 1$ streams guarantees the child obtains a toy with $O(\log^{(k)} n + k)$ queries in expectation.*

Proof. For the i^{th} stream and $s = 1, \dots, n$, let $q_{i,s} = 1$ if the s^{th} box in the stream is empty; otherwise, $q_{i,s} = 0$ to indicate that the box contains a toy. Each $q_{i,s}$ value is decided arbitrarily by the adversary, but independently of the child's choice regarding whether to open the s^{th} box. We calculate the expected number of opened boxes as:

$$\begin{aligned}
&\leq \frac{\ln^{(k)} n}{\delta} + \sum_{i=k-1}^1 \prod_{s=1}^n \left(1 - \frac{\ln^{(i+1)} n}{\delta n} \cdot q_{i,s} \right) \cdot \frac{\ln^{(i)} n}{\delta} \\
&+ \prod_{s=1}^n \left(1 - \frac{\ln n}{\delta n} \cdot q_{0,s} \right) n \\
&\leq \frac{\ln^{(k)} n}{\delta} + \left(\sum_{i=k-1}^1 e^{-\frac{\ln^{(i+1)} n}{\delta n} \sum_{s=1}^n q_{i,s}} \frac{\ln^{(i)} n}{\delta} \right) + e^{-\frac{\ln n}{\delta n} \sum_{s=1}^n q_{0,s}} \cdot n \\
&= \frac{\ln^{(k)} n}{\delta} + \left(\sum_{i=k-1}^1 e^{-\ln^{(i+1)} n} \cdot \frac{\ln^{(i)} n}{\delta} \right) + e^{-\ln n} \cdot n \text{ as } \sum_{s=1}^n q_{i,s} = \delta n \\
&= O(\ln^{(k)} n + k)
\end{aligned}$$

□

The same lower bounds apply to the Adaptive Bad Santa problem. Throughout the remainder of this chapter, the use of a Bad Santa algorithm is used as a subroutine for our reliable broadcast protocols. In this context, an empty box corresponds to a faulty node. By using our Adaptive Bad Santa algorithms, we can tolerate an adversary that selects the locations of its faulty nodes on the fly (but still, independently of the choices by correct nodes). Throughout, for ease of exposition, we do not specify the use of our adaptive variants; however, these could be used if desired.

Error Tolerance in the Bad Santa Protocols and in Reliable Broadcast Protocols: Before describing our reliable broadcast protocols, we first address a possible point of confusion: Previous work under the Byzantine fault model assumed $t < n/2$ whereas in this work we are allowing $t \leq n/2$ in our algorithms for the single-stream Bad Santa problem. This should not be construed as contradicting the lower bound proved in [90]. In order to perform reliable broadcast $t < n/2$ must indeed hold true and, as we shall see in Section 4.4.5, this needs to be the case for Stage 1 of our protocol

in order to propagate the fingerprint. However, in Stage 2, the set G_p can hold $t \leq n/2$ faulty nodes due to our results on the single-stream Bad Santa problem.

Reliable Broadcast Along a Corridor: The presentation of our protocols is limited to demonstrating reliable broadcast along a corridor of width $2r+1$ moving along the positive y -coordinates. That is, we show reliable broadcast for a node $p(x, y)$ where $-r \leq x \leq r$ and $y \geq 0$. This greatly simplifies the description of our results. Furthermore, it is easy to see that reliable broadcast is possible along other corridors traversing the x -coordinates or negative y -coordinates using a synchronization of sending and listening similar to what we describe. The grid can be covered piece-wise with such rectilinear corridors in a number of ways; for example, a spiral suffices (see Figure 4.1). Alternatively, corridors can be appended in many other ways in order to achieve propagation of a message depending on scenario in question. In any event, proving reliable broadcast for this corridor is sufficient to prove reliable broadcast for the grid in general.

4.4.5 Energy-Efficient Reliable Broadcast with Optimal Tolerance

We describe our reliable broadcast protocol that tolerates fail-stop faults; the proof is deferred until the end of Section 4.4.5 since it is subsumed by the proof for the Byzantine case. The pseudocode below shows how broadcast can be achieved along a corridor of width $2r+1$, where $-r \leq x \leq r$, moving along the positive y -coordinates. As mentioned earlier, restricting the movement in this way greatly simplifies our presentation without sacrificing completeness.

Protocol for Fail-Stop Faults

We assume that the nodes in the network know the time slot when the source node will broadcast a message. We will let t_{start} denote the time slot at which the source sends out a message m . The source node located at $(0,0)$ broadcasts m at time slot t_{start} and all correct nodes in $N(0,0)$ are assumed to receive m from the source and commit internally. Nodes in $N(0,0)$ then

broadcast that they have committed to m for the next $2r$ consecutive rounds during their respective allotted time slots.

In the context of our discussion from Section 2.1, we now describe how each node $p(x, y)$, for $-r \leq x \leq r$ and $y \geq r + 1$, listens for and sends messages and, finally, how it broadcasts its committal. Let t_q denote the time slot when q is scheduled to broadcast in round $t_{start} + 2(y - r)$. Using t_q values, each node p creates an ordered set $S_p \subset G_p$ where the elements of S_p are chosen according to the $(1, O(\sqrt{n}))$ strategy for the Bad Santa problem. Node p then awakens from the energy-efficient sleep mode and listens (in order) to nodes in S_p in round $t_{start} + 2(y - r)$. If at any point, p receives a message, it commits to this message internally. During the course of the protocol, node p also facilitates the passage of messages along the two-hop paths. While node p has not committed internally, p listens to each sister node $u(x'', y'')$ in round $t_{start} + 2(y'' - r) + 1$. If p receives a message, then p does the following: (1) commits internally to this message and (2) during its assigned slots p broadcasts m for $2r$ consecutive rounds starting at round $t_{start} + 2(y'' - r) + 2$. Finally, in terms of sending, if at any time a node $p(x, y)$ has committed internally to a message in round $t_{start} + 2(y - r)$ (i.e. used the Bad Santa protocol to commit), p waits until round $t_{start} + 2(y - r) + 1$ and then broadcasts its message for $2r$ consecutive rounds during its assigned time slots. Again, note that in the following pseudocode, each node $p(x, y)$ is such that $-r \leq x \leq r$ and $y \geq 0$.

Finally, we note that our protocol works not only in the case of fail-stop faults, but for slightly more general failure models where a non-malicious faulty sender might send messages that easily recognizable as being incorrect. In such a “fail-safe system”, this kind of value failure is still tolerable by our protocol.

Protocol for Byzantine Faults

Our protocol for the Byzantine fault model runs in two stages. In the first stage, the source propagates a fingerprint $f(m)$ of the message m it wants to broadcast. This fingerprint is assumed to be of size at least $\lg^2 |m|$ bits. Propagation of $f(m)$ is again done using the algorithm in [22]. The second stage is very similar to the previous protocol for the fail-stop faults. In the second stage, the source broadcasts the message m at time slot t_{start} and all correct nodes in $N(0, 0)$ are assumed to receive m from the source and commit

internally. Each node $q(x', y') \in N(0, 0)$ then broadcasts its commitment to m over the next $2r$ consecutive rounds. A node p listens to messages from a set G_p just as in the protocol for fail-stop model. The difference occurs when, at any point a message m' is received. Node p then checks $f(m')$ against the fingerprint f_{maj} to which it committed in the first stage. If they match, p commits to m' internally and executes the broadcast instructions mentioned previously.

We assume that the network alternates between the first stage, where nodes are constantly awake, and the second stage, where nodes are achieving significant energy savings. For instance, it is plausible that internal software could synchronize periodic change-overs between these two stages in much the same way that sensor network alternate periodically between sleep and fully active states to conserve energy in practice. These details are outside the scope of our work and we do not discuss them further.

Finally, note that a faulty node might broadcast an incorrect message m' such that $|m'| > |m|$ where m is the correct message. To avoid complications, we assume that nodes in the network know the size of m and, therefore, can stop listening after receiving $|m|$ bits. For instance, this could be implemented by having the source broadcast the message size in the first stage or having a predefined upper limit on messages size. The details of such solutions would be dictated by context and we omit further discussion of this issue.

We now establish the following preliminary lemma which we will need for our protocols. For a source located at (x, y) , label the set of nodes in the corridor as $S_{cor} = S_{x,cor} \cup S_{y,cor}$ where $S_{x,cor} = \{q(x', y') \mid (r + 1 \leq x' \leq x) \wedge (y - r \leq y' \leq y + r)\}$ and $S_{y,cor} = \{q(x', y') \mid (-r \leq x' \leq r) \wedge (0 \leq y' \leq y + 2r)\}$. The coordinate (x, y) will always be clear from the context; indeed, in the context of our proofs, we only deal with the location of the source with $x = 0$ and $y = 0$. Figure 4.2(a) illustrates a corridor for $r = 3$. Finally, recall that a *round* is one iteration through the broadcast schedule.

$(1, O(\sqrt{n}))$ RELIABLE BROADCAST FOR THE FAIL-STOP FAULT MODEL

1. At time slot t_{start} , the source $d(0, 0)$ does a one-time local broadcast of m and each node in $N(d)$ commits internally to m .
2. All nodes in $N(0, 0)$ broadcast their commitment to m for the next consecutive $2r$ rounds.

The following portion of the protocol is followed by all nodes not in $N(0, 0)$:

3. If node $p(x, y)$ has committed internally to a message in round $t_{start} + 2(y - r)$ (i.e. in Step 5), then p waits until round $t_{start} + 2(y - r) + 1$ and then broadcasts its message for $2r$ consecutive rounds during its assigned time slots.
4. While node $p(x, y)$ has not committed internally to a message, node p listens to each sister node $u(x'', y'')$ in round $t_{start} + 2(y'' - r) + 1$. If p receives the message m from u , then p does the following: (1) commits internally m and (2) during its assigned slots p broadcasts m for $2r$ consecutive rounds starting at round $t_{start} + 2(y'' - r) + 2$.
5. While node $p(x, y)$ has not committed internally to a message, p does the following. For a node $q \in G_p$, let t_q denote the time slot when q is scheduled to broadcast in round $t_{start} + 2(y - r)$. Using t_q values, node p creates an ordered set $S_p \subset G_p$ where the elements of S_p are chosen according to the $(1, \sqrt{n})$ Bad Santa strategy. Then p does the following:
 - Node $p(x, y)$ listens to $q \in S_p$ in round $t_{start} + 2(y - r)$. If at any point p receives a message m , then p commits to m internally, breaks the for-loop and proceeds to Step 3.

The following Lemma 22 is useful for our Byzantine-tolerant protocols. In particular, it provides an analysis of the previous protocol in [22] with the minor modification that a node waits for the (at most) two messages from its sister nodes before issuing HEARD messages. We then later use this

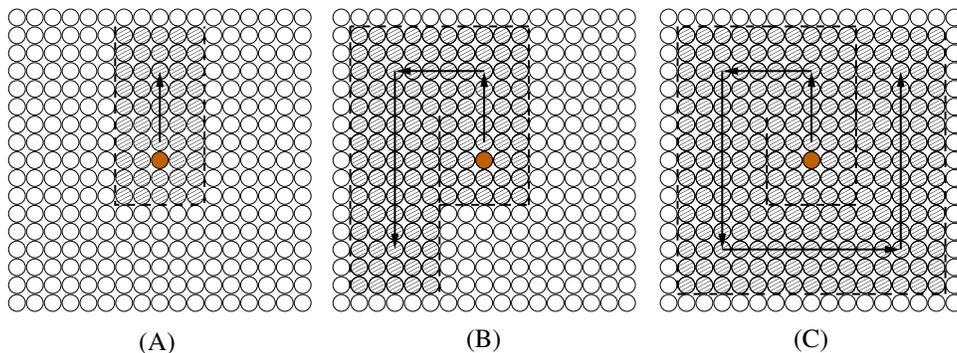


Figure 4.1: The source is denoted by the shaded node which has location $(0, 0)$. (A) Movement in the positive y direction establishing a $(2r + 1) \times (2r + 1)$ square of committed nodes above $N(0, 0)$; this is the corridor we explicitly address in the proofs of correctness for our protocols. (B) Spiraling out from $N(0, 0)$, movement in the negative x direction and then the negative y direction. (C) Further depiction of the spiral expansion of committed nodes along a corridor of width $2r + 1$.

result to address the necessary delay between Stage 1, where a fingerprint is propagated, and Stage 2 of our protocol, when the full message is sent. Note that in Lemma 22, we deal with arbitrary x and y values.

Lemma 22. *Assume a broadcast schedule where no collisions occur and each node can broadcast once every round as discussed in Section 4.1.1. Consider a source, $d(0, 0)$, that broadcasts a fingerprint $f(m)$ at time slot t_0 under either the fail-stop or Byzantine fault models where $t < \frac{r}{2}(2r + 1)$. Then by using the protocol of [22], node $p(x, y)$ is able to commit to $f(m)$ by round $t_0 + 2(|x| + |y|)$.*

Proof. We are essentially following the argument for correctness given in [22] and discussed in Section 2.1; however, we are restricting our view to those nodes in S_{cor} . That is, nodes in S_{cor} will only accept messages from other nodes in S_{cor} and they will ignore all messages they receive from nodes outside the corridor. Clearly, this can only result in a slowdown in the propagation of the broadcast value; moreover, the rectilinear shape of the corridor can only slow down the rate of propagation in comparison to the original propagation described in [22]. An argument identical to that in [22] can be used to show that each correct node $q(x', y') \in S_{cor}$ will commit to the correct fingerprint

by receiving messages along at least $2t + 1$ node disjoint paths of the form (u_i, q) and (u_i, u'_i, q) as shown in Figure 4.2(a). While we do not repeat the entire argument here, Figure 4.2(b) illustrates the set G_p for each node p in a row of the corridor along increasing y -values. That is, the regions A_p , B_p and B'_p are illustrated for each position in the context of the proof discussed in Section 2.1.

We now consider the time required until $p(x, y)$ can commit to $f(m)$ regardless of which nodes in the corridor fail; p does so by listening to the nodes in G_p . Without loss of generality, assume that x, y are positive coordinates and that the broadcast first moves along nodes in $S_{y,cor}$ (moving up) and then along nodes in $S_{x,cor}$ (moving right). At t_0 , the source broadcasts $f(m)$ and all nodes in $N(0, 0)$ commit to $f(m)$. Consider a node $q(a, r + 1)$ where $-r \leq a \leq r$. It takes at most one round for q to receive messages along paths of the form (u_i, q) from region A . Concurrently, in this one round, nodes u_i can transmit messages to nodes u'_i along paths of the form (u_i, u'_i, q) (region B to B') where the HEARD messages from the (at most) two sister nodes are appended in a single message. At most an additional round is required to send from nodes u_i to q . Therefore, at most two rounds are required before q can commit. Note that this holds for all nodes with coordinates $(a, r + 1)$ for $-r \leq a \leq r$; this entire row can commit after at most two rounds. It follows that all nodes up to and including row y in $S_{y,cor}$ are committed to $f(m)$ after $t_0 + 2(y + r)$ rounds; the remaining r rows in $S_{y,cor}$ do not commit. An identical argument shows that all nodes in $S_{x,cor}$ are committed to $f(m)$ after $t_0 + 2(x - r)$ rounds. Therefore, p commits after at most $t_0 + 2(x + y)$ rounds; if x and y can take on negative values, this becomes $t_0 + 2(|x| + |y|)$. \square

$(1, O(\sqrt{n}))$ RELIABLE BROADCAST FOR THE BYZANTINE FAULT MODEL

Stage 1:

1. At time t_0 , the source uses the reliable broadcast protocol of [22] to broadcast the fingerprint $f(m)$ to all nodes in the grid.

Stage 2:

2. At time slot t_{start} , the source $d(0, 0)$ does a one-time local broadcast of m and each node in $N(d)$ commits internally to m .

3. All nodes in $N(0, 0)$ broadcast their committal to m for the next consecutive $2r$ rounds.

The following portion of the protocol is followed by all nodes not in $N(0, 0)$:

4. If node $p(x, y)$ has committed internally to a message in round $t_{start} + 2(y - r)$ (i.e. in Step 6), p waits until round $t_{start} + 2(y - r) + 1$ and then broadcasts its message for $2r$ consecutive rounds during its assigned time slots.
5. While node $p(x, y)$ has not committed internally to a message, node p listens to each sister node $u(x'', y'')$ in round $t_{start} + 2(y'' - r) + 1$. If the message m_u that p receives from u equals the majority fingerprint value, f_{maj} , then p does the following: (1) commits internally m_u and (2) during its assigned slots p broadcasts m_u for $2r$ consecutive rounds starting at round $t_{start} + 2(y'' - r) + 2$.
6. While node $p(x, y)$ has not committed internally to a message, p does the following. For a node $q \in G_p$, let t_q denote the time slot when q is scheduled to broadcast in round $t_{start} + 2(y - r)$. Using t_q values, node p creates an ordered set $S_p \subset G_p$ where the elements of S_p are chosen according to the $(1, \sqrt{n})$ Bad Santa strategy. Then p does the following:
 - Node $p(x, y)$ listens to $q \in S_p$ in round $t_{start} + 2(y - r)$. In listening to each q , p will obtain a value m_q (or nothing, if q is Byzantine and sends nothing). If at any point $f(m_q)$ equals the f_{maj} value of p , then p commits to m_q internally, breaks the for-loop and proceeds to Step 4.

The next lemma proves that, if we assume that the adversary cannot cause a collision with f_{maj} , then each node can commit to the correct message using our protocol. In particular, it establishes that the second stage of our protocol achieves the $2t + 1$ connectedness necessary for reliable broadcast. The lemma also establishes that the broadcasting and receiving actions by each node are correct. Finally, the resource costs per node for Stage 2 follow immediately. Note that this stops short of proving Theorem 9 since the issue of fingerprints has not yet been addressed. While we include it for completeness, we stress that the $2t + 1$ connectedness component of the proof is essentially an adaptation of the proof found in [22] which was reviewed in the beginning of Section 2.1. Again, the proof focuses on movement along the positive y -coordinates along a corridor of width $2r + 1$ where $-r \leq x \leq r$. Figure 4.3 illustrates how our protocol proceeds when $r = 3$.

Lemma 23. *Assume a broadcast schedule where no collisions occur and each node can broadcast once every round as discussed in Section 4.1.1. Furthermore, assume each node already possesses f_{maj} prior to receiving any other messages and that if a message m received by a correct node p corresponds to the fingerprint f_{maj} propagated in the first step, then m is the correct message. Under these assumptions, the $(1, O(\sqrt{n}))$ Reliable Broadcast for the Byzantine Fault Model protocol has the following properties:*

- *Each node $p(x, y)$ where $-r \leq x \leq r$ (except those on the boundary of width r if the grid is finite) commits to the correct message m by round $\max\{2(y - r), 0\}$.*
- *Each node is awake for $O(\sqrt{n})$ time slots in expectation. Each node sends and receives $O(\sqrt{n}|m|)$ bits in expectation.*

Proof. For simplicity, we normalize such that t_{start} is round 0. Our proof is by induction and throughout we assume that each node has an x -coordinate such that $-r \leq x \leq r$:

Base Case: Each node in $N(0, 0)$ commits to the correct message m immediately upon hearing it directly from the dealer. Therefore, each node $p(x, y) \in N(0, 0)$ commits to m by round $0 \leq \max\{2(y - r), 0\}$.

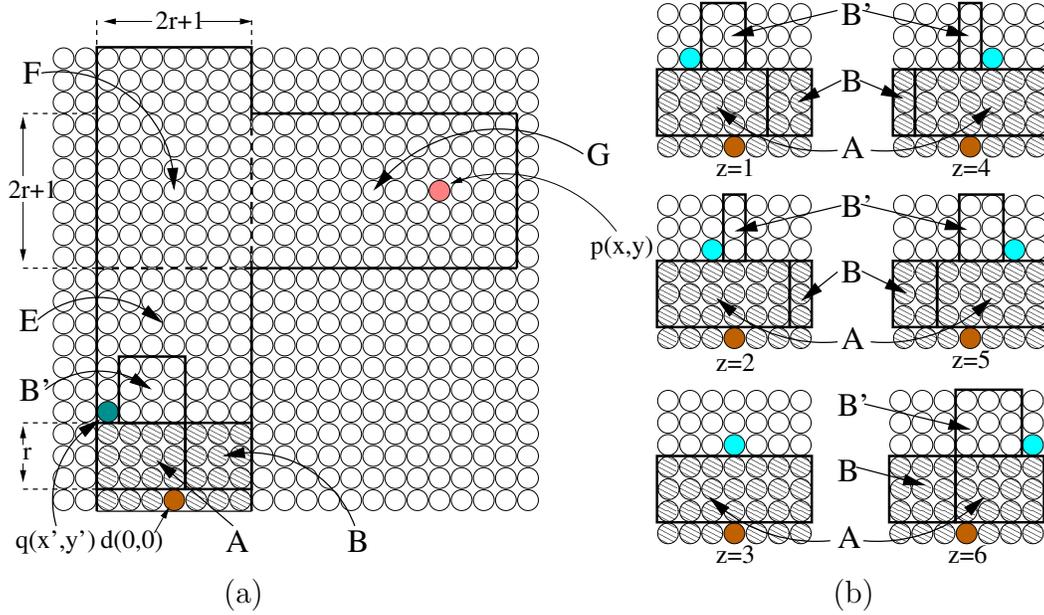


Figure 4.2: Let p be a node that is not at the boundary of width r in the grid. (a) A depiction of a corridor for $r = 3$. Together the nodes in region E and F constitute $S_{y,cor}$ while the nodes in G constitute $S_{x,cor}$. Node disjoint paths of the form (u_i, q) originate from nodes u_i in region A . As discussed in [21, 22], node disjoint paths of the form (u_i, u'_i, q) originate from nodes u_i in region B and traverse through nodes u'_i in B' to reach node q . (b) The regions A , B and B' are illustrated for each node along a row of $S_{y,cor}$. The value for z is given for each position in the context of the proof reviewed in Section 2.1.

Induction Hypothesis: For simplicity, we will assume as before that $p \in N(a, b + 1)$ where $-r \leq a \leq r$; the other cases for proving the statement for $p \in (a, b)$ follow by symmetry. In this context, the induction hypothesis is as follows: if each $p'(x', y') \in N(a, b)$ has committed to m by round $2(y' - r)$, then each correct node $p(x, y) \in N(a, b + 1) - N(a, b)$ is able to commit to m by round $2(y - r)$.

Induction Step: As we reviewed before in the beginning of Section 2.1 of Chapter 2, we show $2t + 1$ connectedness in a single neighborhood. We will

argue simultaneously about the time required for p to hear messages along these disjoint paths. The node $p(x, y)$ lies in $N(a, b + 1) - N(a, b)$ and can be considered to have location $(a - r + z, b + r + 1)$ where $0 \leq z \leq r$ (the case for $r + 1 \leq z \leq 2r$ follows by symmetry). We demonstrate that there exist $r(2r + 1)$ node-disjoint paths $P_1, \dots, P_{r(2r+1)}$ all lying within the same neighborhood and that the synchronization prescribed by our protocol is correct:

- **One-Hop Paths:** the set of nodes $A_p = \{q(x, y) \mid (a - r) \leq x \leq (a + z) \text{ and } (b + 1) \leq y \leq (b + r)\}$ lie in $N(a, b)$ and are neighbors of p . Therefore, there are $r(r + z + 1)$ paths of the form $q \rightarrow p$ where $q \in A_p$.

By their position relative to $p(x, y)$, each correct node $q(x', y') \in A_p$ is such that $y - r \leq y' \leq y - 1$. Therefore, by the induction hypothesis, a correct node $q \in A_p$ commits in round $2(y - 2r)$ at the earliest and $2(y - r - 1)$ at the latest. Consequently, a correct node in A_p starts broadcasting its committals in round $2(y - 2r) + 1$ at the earliest and $2(y - r - 1) + 1$ at the latest. In the former case, recall that broadcasting occurs for $2r$ rounds, which means that q is broadcasting from round $2(y - 2r) + 1$ to $2(y - r)$, inclusive, at the earliest. In the latter case, q is broadcasting from $2(y - r - 1) + 1$ to $2(y - 1)$, inclusive. Therefore, all correct nodes in A_p are broadcasting a committal message in round $2(y - r)$ and so $p(x, y)$ can receive a message from each correct node in A_p in this round.

- **Two-Hop Paths:** consider the sets $B_p = \{q(x, y) \mid (a + z + 1) \leq x \leq (a + r) \text{ and } (b + 1) \leq y \leq (b + r)\}$ and $B'_p = \{q'(x, y) \mid (a + z + 1 - r) \leq x \leq (a) \text{ and } (b + r + 1) \leq y \leq (b + 2r)\}$. The nodes in B_p lie in $N(a, b)$ while the nodes in B'_p lie in $N(p)$. Moreover, the set B'_p is obtained by shifting left by r units and up by r units. Recall that there is a one-to-one mapping between the nodes in B_p and the nodes in B'_p ; these are sister nodes. There are $r(r - z)$ paths of the form $q \rightarrow q' \rightarrow p$.

Consider a correct node $q(x', y') \in B_p$ and its sister node $q'(x'', y'') \in B'_p$. Again, given the location of $q(x', y')$ relative to $p(x, y)$, by the induction hypothesis, the earliest $q \in N(a, b)$ has committed is $2(y - 2r)$ and the latest is $2(y - r - 1)$. Therefore, by protocol, q starts broadcasting its committal $2r$ times starting in round $2(y - 2r) + 1$ at the earliest and $2(y - r - 1) + 1$ at the latest. The sister node

of $q, q' \in B'_p$, listens to q in the first round that q broadcasts. If q' receives a correct m , then q' broadcasts this $2r$ times; therefore, this starts in round $2(y - 2r) + 2 = 2(y - 2r + 1)$ at the earliest and $2(y - r - 1) + 2 = 2(y - r)$ at the latest. In the former case, recall that q' broadcasts for $2r$ consecutive rounds and therefore is broadcasting until round $2(y - r + 1) - 1 > 2(y - r)$. Therefore, all correct nodes in B'_p with a message to broadcast are doing so in round $2(y - r)$ and so $p(x, y)$ can hear a message from any such $q' \in B'_p$ in this round.

Therefore, there are a total of $r(r + z + 1) + r(r - z) = r(2r + 1)$ node-disjoint paths from $N(a, b)$ to $PN(a, b)$, all lying in a single neighborhood $N(a, b + r + 1)$. By our argument above, each correct node $p(x, y)$ receives the one-hop and two-hop messages over these paths by round $2(y - r)$. We note that (1) more than half of these paths will provide the correct message and (2) the sampling follows the Bad Santa protocol which is a Las Vegas algorithm. Therefore, we are guaranteed that p will obtain a message m that corresponds to f_{maj} . Finally, by our initial assumption regarding the inability of the adversary to forge a collision, this means that m is the correct message.

We now analyze the resource bounds for our protocol. Consider the situations where p must deal with (either broadcasting or receiving) a message: (1) p receives messages in order to commit, (2) p broadcasts it has committed, and (3) p facilitates two-hop messages. We consider each case. To address (1), note that p uses the Bad Santa protocol; while in the streaming problem, we attempt to obtain a 1 at unit cost per query, here node p is attempting to select a correct node at the cost of listening to $|m|$ bits per selection.³ This method of sampling from G_p means p receives $O(\sqrt{n})$ messages in expectation. To address (2), note that p broadcasts that it has committed $2r = O(\sqrt{n})$ times. To address (3), we consider $p \in PN(a, b + 1)$ as before, and note that p belongs to many B'_q sets for different nodes q ; however, regardless of which B'_q set, p only ever has two sister nodes. Therefore, considering broadcast along the x and y coordinates, the number of sister nodes is $O(1)$; the number of broadcasts due to two-hop paths is thus $O(r)$. In conclusion (not counting the fingerprint, since we are dealing only with

³Selecting a random node is necessary; if not, the adversary might have faulty nodes send correct fingerprints in the first round and, if p selects nodes from G_p in a deterministic fashion, the adversary may force p to listen to many messages that do not hash to f_{maj} .

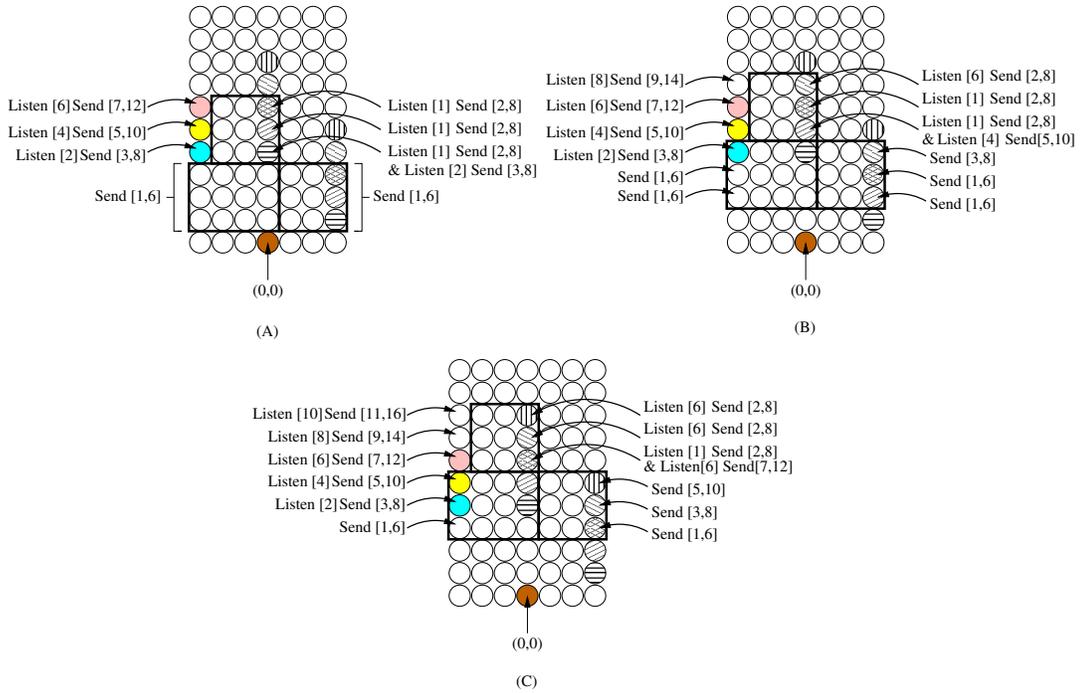


Figure 4.3: A depiction of the $(1, O(\sqrt{n}))$ Reliable Broadcast for the Byzantine Fault Model protocol for $r = 3$. Times for broadcasting and receiving are denoted by $[a, b]$ which denotes rounds a through b inclusive. (A) Shows how a node p in row 4 can commit by listening to nodes in $G_p = A_p \cup B'_p$; we focus on nodes on the left-most edge. Note the node in row 4 marked with horizontal lines. This node acts as part of B'_p while also committing as other nodes in row 4 do; later, it will act as a node in B'_i and A_j for other nodes i and j by sending in rounds $[3, 8]$. Note that this node is sending from round 2 to 8 (inclusive) but we separate this into $[2, 8]$ and $[3, 8]$ to make the different roles explicit. (B) and (C) Depictions of the timing of broadcasting and receiving as nodes in rows 5 and 6 commit, respectively.

the second stage of our protocol) each node is awake for $O(\sqrt{n})$ time slots in expectation, sends $O(\sqrt{n}|m|)$ bits and receives $O(\sqrt{n}|m|)$ bits. \square

With Lemma 22 and Lemma 23 in hand, we can now give the proof for Theorem 4:

Proof. We begin by proving correctness and we start with Stage 1. Stage 1 of the protocol is no different than the broadcast presented in [22] where the value being transmitted is a fingerprint. Consequently, every correct node will be able to derive a majority fingerprint f_{maj} .

We now analyze Stage 2. Lemma 23 assumes that (1) we have an appropriate schedule, (2) each node has f_{maj} prior to receiving any other messages, and (3) if m corresponds to f_{maj} then m is correct. We go about addressing these three criteria:

- First, we can assume the schedule of [90] which satisfies the properties required by Lemma 23.
- Second, consider all nodes in a square of size $3(2r+1) \times 3(2r+1)$ centered about $(0, 0)$; the node at the top-right has position $(3r+1, 3r+1)$. By selecting t_{start} at least $2(6r+2)$ rounds after the time of sending of the fingerprint, t_0 (that is $t_{start} \geq t_0 + 2(6r+2)$), then Lemma 22 guarantees that by time t_{start} , all nodes in a square of size $3(2r+1) \times 3(2r+1)$ centered about $(0, 0)$ will have committed to the fingerprint. If we assume, as mentioned earlier prior to presenting the pseudocode, that the message expands via a spiral corridor of width $2r+1$ from $N(0, 0)$, then this guarantees that the propagation of the fingerprint will always be sufficiently far ahead of the propagation of the full message to allow nodes to first commit to the fingerprint. Note that if m is propagated in a different fashion (i.e. not a spiral) then the timing offset would need to be adjusted accordingly.
- Third, by assumption, f is a secure hash function and the size of the fingerprint is $\lg^2 |m|$. Therefore, given $f(x)$, the probability that the adversary obtains a value $x' \neq x$ such that $f(x') = f(x)$ is $2^{-\lg^2 |m|} = |m|^{-\lg |m|}$. It will take the adversary superpolynomial time in $|m|$ to forge such an x' and so f_{maj} will correspond to the correct value m . Recall that s is the number of computational steps afforded to the adversary; i.e. the number of times the adversary can create an input x' , apply f to x' and check for a match between the output fingerprint

$f(x')$ and f_{maj} . Therefore, given that p receives a message from a correct node in G_p that when hashed matches the fingerprint to which p committed, with error $O(s/|m|^{\lg |m|})$, this message is the correct message sent by the source where s is the number of computational steps available to the adversary.

Finally, we analyze resource costs. Lemma 23 confirms the amount of awake time specified by Theorem 9 after the sending of the fingerprint. Regarding the bit complexity over both Stages 1 and 2, we need to consider the additional cost due to sending the fingerprint. Each node p broadcasts and receives $O(r^2) = O(n)$ fingerprints, for a total of $O(n \log^2 |m|)$ bits in Stage 1. Therefore, the expected bit complexity over both stages is $O(n \log^2 |m| + \sqrt{n}|m|)$. \square

The above proof essentially subsumes the proof for Theorem 3; however, we include it here for completeness:

Proof. The proof of correctness for the fail-stop model differs in two places from the proof of Theorem 4. For criteria (2), there is no need for a fingerprint. For criteria (3), since messages are never corrupted, only lost if a fault occurs, p is guaranteed that the message it receives from a correct $q \in G_p$ is correct. Finally, for the fail-stop model, the resource costs are easy to analyze. The awake times due to listening follow directly from the fact that each node broadcasts $O(\sqrt{n})$ times and uses the Bad Santa problem for listening; therefore, a total of $O(\sqrt{n})$ time slots in expectation. In terms of bit complexity, each node p broadcasts $|m|$ for r rounds times and listens to m once. Therefore, p broadcasts $O(\sqrt{n}|m|)$ bits and receives $|m|$ bits. \square

It may seem that, with some modifications to the protocol, we can employ a multi-stream Bad Santa strategy to achieve further expected savings. We now explain why this is not the case. Note that such a change would require each node to send $O(r \cdot k \cdot |m|)$ bits while reducing the expected listening cost to $O(k \cdot |m|)$. However, since the costs for sending and receiving are of the same magnitude, we do not achieve an overall asymptotic savings when we consider the addition of these two communication costs. That is, the $O(r \cdot k \cdot |m| + k \cdot |m|)$ expected cost we would obtain with the multi-stream Bad Santa strategy is no better (asymptotically) than the $O(r|m|)$ expected cost we already achieve with the single-stream Bad Santa strategy.

The impediment here is that, in order to tolerate the optimal number of faults, the propagation of m must proceed incrementally row-by-row up the corridor; this necessitates the r factor that appears in our analysis. We will see in the next section that, if we incur a small decrease in the number of faults we can tolerate, we can achieve further energy savings.

Finally, we comment on the difference in running time between our algorithms for the fail-stop and Byzantine fault models. Clearly, the need to propagate a fingerprint in the Byzantine case incurs additional time. However, as we have seen, the dealer need wait only $2(6r+2)$ rounds after sending the fingerprint before broadcasting the full message.

4.4.6 Energy-Efficient Reliable Broadcast with Near-Optimal Fault Tolerance

When $t \leq (1-\epsilon)\frac{r}{2}(2r+1)$ for any constant $\epsilon > 0$, we show how to achieve even larger energy savings by employing the $(k+1, O(\log^{(k)}(n/2) + k))$ strategy to the Bad Santa problem for $k \geq 1$. As we will show, a correct node may listen to at least $(r/2)(2r+1)$ messages, of which a $(1-\epsilon)$ -fraction may be faulty; therefore, we are now allowing more than a $1/2$ fraction of paths to deliver faulty messages. We point out that, in actuality, our results hold for $t \leq (1-\epsilon)(1+r+r^2)$ which is larger than $(1-\epsilon)(r/2)(2r+1)$ by an amount of $(1-\epsilon)(1+r/2)$. However, asymptotically, this difference is negligible and we phrase the result in this manner to illustrate that we are within an arbitrary constant fraction of the optimal tolerance.

In this case, we present a Byzantine fault-tolerant reliable broadcast protocol. The protocol is very similar to our $(1, O(\sqrt{n}))$ Reliable Broadcast for the Byzantine Fault Model presented in Section 4.4.5 where here we use the $(k+1, O(\log^{(k)}(n) + k))$ strategy; however, there are important distinctions. In particular, the sets A_p , B'_p and B_p are defined in a slightly different manner in the correctness proof for our protocol later on. Furthermore, each node broadcasts for $O(k)$ (rather than r) consecutive rounds and the synchronization of broadcasting and receiving is altered. Essentially, r rows of a corridor are committing every $k+2$ rounds; this is different from the previous protocols where each row committed in a different round. The pseudocode is given below and, again, deals with movement along the positive y -coordinates.

$(k + 1, O(\log^{(k)}(n) + k))$ RELIABLE BROADCAST FOR THE BYZANTINE FAULT MODEL

Stage 1:

1. At time slot t_0 , the source uses the reliable broadcast protocol of [22] to broadcast the fingerprint $f(m)$ to all nodes in the grid.

Stage 2:

2. At time slot t_{start} , the source $d(0, 0)$ does a one-time local broadcast of m and each node in $N(d)$ commits internally to m .
3. All nodes in $N(0, 0)$ broadcast their committal to m for the next consecutive $k + 2$ rounds.

The following is followed by all nodes not in $N(0, 0)$:

4. If node $p(x, y)$ has committed internally to a message via listening to a set $S_{p,i}$ for $i = 0, \dots, k$ (i.e. Step 6), node p uses its allotted time slot to broadcast this fact for $k + 2$ consecutive rounds; that is, from round $(k + 2) \left(\lfloor \frac{y-1}{r} \rfloor \right) + 1$ to $(k + 2) \left(\lfloor \frac{y-1}{r} \rfloor + 1 \right)$ inclusive.
5. While node $p(x, y)$ has not committed to a message, node p listens to each sister nodes in round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor \right) + 1$. If the message m_u that p receives from u equals the f_{maj} value, then p does the following: (1) commits internally m_u and (2) during its assigned slots p broadcasts m_u for $k + 1$ consecutive rounds: from round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor \right) + 2$ to round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor + 1 \right)$, inclusive.
6. While node $p(x, y)$ has not committed internally to a message, p does the following. For a node $q \in G_p$, let t_q denote the time slot when q is scheduled to broadcast in round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor \right) + 2$. Using t_q values, node p creates ordered sets $S_{p,0}, \dots, S_{p,k}$ where $S_{p,i} \subset G_p$ for $i = 0, \dots, k$ where the elements of each $S_{p,i}$ are chosen according to the $(k+1, O(\log^{(k)}(n)+k))$ Bad Santa strategy. Then for $i = 0, \dots, k$, p does the following:
 - Node $p(x, y)$ listens to each node $q \in S_{p,i}$ for $k + 1$ consecutive rounds: that is, from round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor \right) + 2$ to round $(k + 2) \left(\lfloor \frac{y-r-1}{r} \rfloor + 1 \right)$. If at any point q receives a message m_q such that $f(m_q)$ equals the f_{maj} value of p , then p commits to m_q internally, breaks the for-loop and proceeds to Step 4.

From the point of view of each correct node, the streams are arriving consecutively; there is no interleaving of the streams. The protocol is involved, and to avoid possible confusion, we draw attention to the fact that nodes acting as members of A_p sets broadcast for $k+2$ times, even though the corresponding Bad Santa protocol uses $k+1$ streams. This is because of the extra delay of one round incurred by the two-hop messages; we note that nodes in B'_p sets that facilitate these messages broadcast for $k+1$ consecutive rounds. The correctness of this protocol can be demonstrated in a similar fashion to the preceding protocols; however, there is a difference in that now the proof deals with all nodes in r rows rather than a single row. Figure 4.4 illustrates how this protocol proceeds when $r = 3$ and $k = 3$. For completeness, we establish Theorem 11; we again only consider movement along the positive y -coordinates.

Proof. The proof is again by induction and, for simplicity, we assume $t_{start} = 0$ and we again assume that each node in the corridor has an x -coordinate such that $-r \leq x \leq r$. We show $2t+1$ connectedness inside a single neighborhood and that each node $p(x, y)$, for $-r \leq x \leq r$, commits by round $(k+2) \left(\max \left\{ \left\lfloor \frac{y-r-1}{r} \right\rfloor + 1, 0 \right\} \right)$.

Base Case: Each node in $N(0, 0)$ commits to the correct message m immediately upon hearing it directly from the dealer; that is, by round 0.

Induction Hypothesis: Rather than dealing with nodes in $p \in N(a, b+1) - N(a, b)$, our proof differs in that we address all nodes in $p \in N(a, b+r) - N(a, b)$ i.e. all nodes in the r rows above row b and for simplicity we will assume $b > 0$ (we do not deal with the nodes in $N(0, 0)$) and that $-r \leq a \leq r$. In particular, the induction hypothesis is as follows: if each node $p'(x', y')$ in rows $b+1, \dots, b+r$ of $N(a, b)$ commit to m in round $(k+2) \left(\left\lfloor \frac{y'-r-1}{r} \right\rfloor + 1 \right)$, then each correct node $p(x, y) \in N(a, b+r) - N(a, b)$ commits to m in round $(k+2) \left(\left\lfloor \frac{y-r-1}{r} \right\rfloor + 1 \right)$.

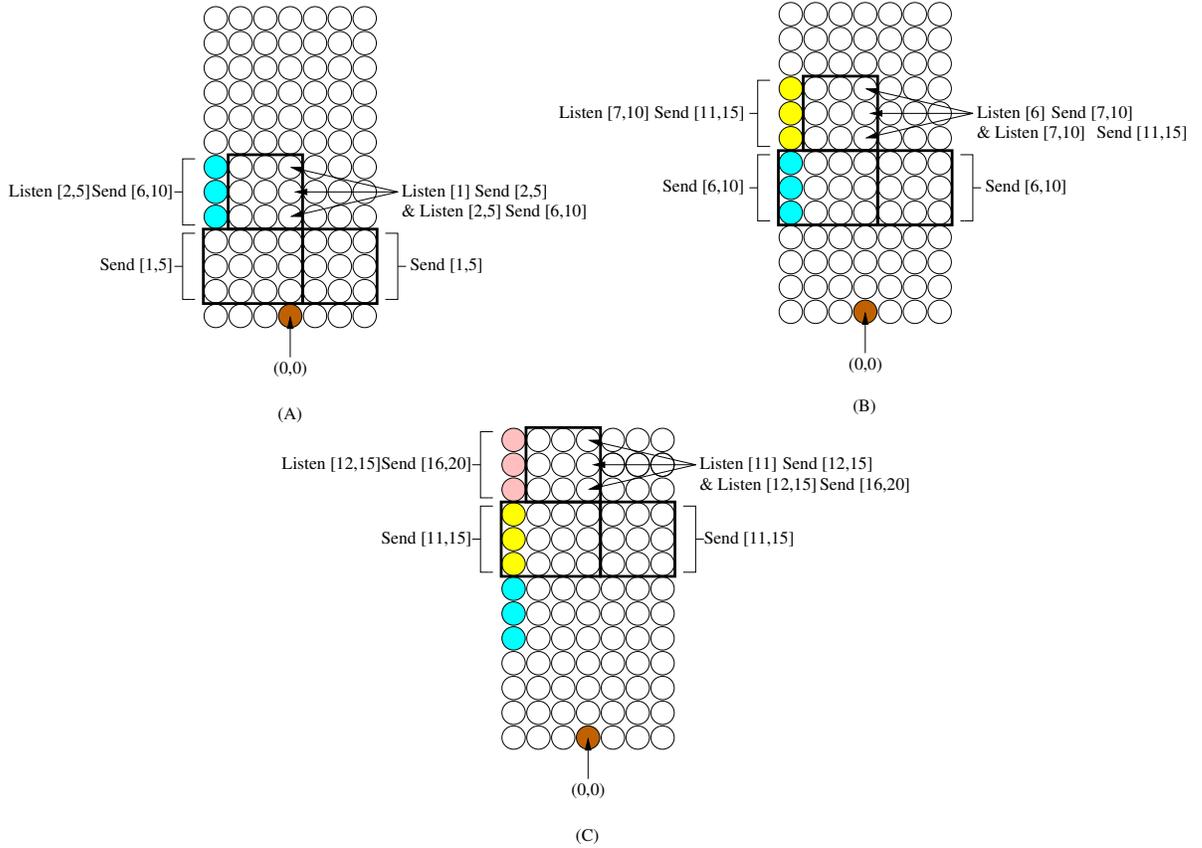


Figure 4.4: A depiction of the $(k + 1, O(\log^{(k)}(n) + k))$ Reliable Broadcast for the Byzantine Fault Model protocol for $r = 3$ and $k = 3$. Times for broadcasting and receiving are denoted by $[a, b]$ which denotes rounds a through b inclusive. (A) Shows how all nodes in rows 4, 5, and 6 commit; we focus on nodes along the leftmost edge. Node p_1 along the edge in row 4 can commit by listening to all nodes in $G_p = A_p \cup B'_p$. In contrast, p_3 in row 6 can sample from only the top row in A_p , which consists of $r + 1$ nodes, and all of B'_p , which consists of r^2 nodes; therefore, p_3 can sample from $1 + r + r^2$ nodes. (B) and (C) Depictions of the timing of broadcasting and receiving as nodes in rows 7, 8, 9 and, subsequently, nodes in rows 10, 11, 12 commit, respectively.

Induction Step: We show $2t + 1$ connectedness and simultaneously prove the correctness of the timing for broadcasting and receiving. The node $p(x, y)$ lies in $N(a, b + r) - N(a, b)$ and can be considered to have location $(a - r + z, b + r + 1 + c)$ where $0 \leq z \leq r$ (the case for $r + 1 \leq z \leq 2r$ follows by symmetry) and $0 \leq c \leq r - 1$. We demonstrate that there exist at least $1 + r + r^2$ node-disjoint paths P_1, \dots, P_{1+r+r^2} all lying within the same neighborhood and that the synchronization prescribed by our protocol is correct. As we mentioned previously, the sets A_p, B'_p and B_p are defined slightly differently than previously; they are defined below in our proof and Figure 4.5 depicts these sets.

- **One-Hop Paths:** the set of nodes $A_p = \{q(x, y) \mid (a - r) \leq x \leq a \text{ and } (b + 1 + c) \leq y \leq (b + r)\}$ lie in $N(a, b + r + 1)$. Therefore, node $p(a - r + z, b + r + 1 + c)$ can receive broadcasts from nodes in at least $r - c \geq 1$ row(s) of A_p which amounts to at least $r + 1$ nodes.

Consider a correct node $q(x', y')$ in the r rows $b + 1, \dots, b + r$ of $N(a, b)$ and recall $p(x, y)$ is in some row $b + r + 1, \dots, b + 2r$. The induction hypothesis guarantees that q has committed by round $(k + 2)(\lfloor \frac{y' - r - 1}{r} \rfloor + 1) = (k + 2)(\lfloor \frac{y - r - 1}{r} \rfloor)$. Then, by the protocol, q broadcasts for $k + 2$ consecutive rounds; that is, from round $(k + 2)(\lfloor \frac{y - r - 1}{r} \rfloor) + 1$ to round $(k + 2)(\lfloor \frac{y - r - 1}{r} \rfloor) + k + 2$, inclusive. Node p is scheduled to begin listening in round $(k + 2)(\lfloor \frac{y - r - 1}{r} \rfloor) + 2$ and so p can receive a message from each such q for $k + 1$ consecutive rounds. Therefore, p hears all one-hop messages by round $(k + 2)(\lfloor \frac{y - r - 1}{r} \rfloor + 1)$

- **Two-Hop Paths:** consider the sets $B_p = \{q(x, y) \mid (a + 1) \leq x \leq (a + r) \text{ and } (b + 1) \leq y \leq (b + r)\}$ and $B'_p = \{q'(x, y) \mid (a - r + z + 1) \leq x \leq (a + z) \text{ and } (b + r + 1 - c) \leq y \leq (b + 2r)\}$. The nodes in B_p form an $r \times r$ square within $N(a, b)$ while the nodes in B'_p , again an $r \times r$ square, lie in the neighborhood $N(a, b + r + 1)$. Note that now the set B'_p is no longer necessarily obtained by shifting left by r units and up by r units; now it is obtained by shifting left by $r - z$ units and up by r units. There is still a one-to-one mapping between the nodes in B_p and the nodes in B'_p ; these are sister nodes. There are r^2 paths of the form $q \rightarrow q' \rightarrow p$.

From the point of view of $p(x, y)$, consider a correct node $q(x', y') \in B_p$. By the induction hypothesis, q in one of the rows $b + 1, \dots, b + r$ of $N(a, b)$

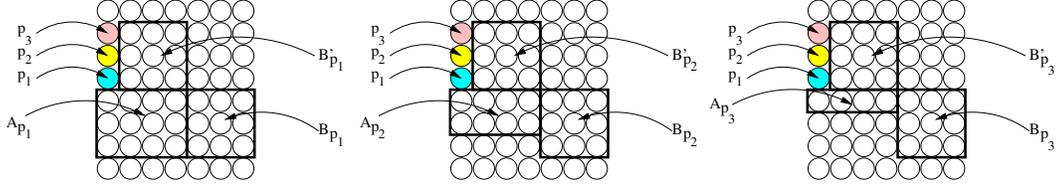


Figure 4.5: The sets A_{p_i} , B'_{p_i} and B_{p_i} for $i = 1, 2, 3$ and $r = 3$. (A) For the node p_1 with a y -coordinate such that $y \bmod r = 1$, the sets are defined the same way. (B) Node p_2 has an A_{p_2} set which consists only of the two top rows of A_{p_1} . (C) Node p_3 has an A_{p_3} set which consists only of the top row of A_{p_1} .

has committed by round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor + 1) = (k+2)(\lfloor \frac{y-r-1}{r} \rfloor)$. By the protocol, its sister node $q' \in B'_p$ listened at the first of these time slots; hence, q' can receive a message from q in round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + 1$. If q' received a correct m , then q' would broadcast m starting in round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + 2$ for $k+1$ consecutive rounds. Therefore, q' broadcasts a correct message from round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + 2$ to round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + k + 2 = (k+2)(\lfloor \frac{y-r-1}{r} \rfloor + 1)$ (inclusive). Node p is scheduled to begin listening in round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + 2$ and so p can receive a message from each such $q' \in B'_p$ for $k+1$ consecutive rounds. Therefore, p hears all two-hop messages by round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor + 1)$

Given our timing analysis above, we point out that, from p 's point of view, the messages from 1-hop and 2-hop paths arrive in the same round for each of the $k+1$ rounds starting at round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor) + 2$; therefore, we have $k+1$ streams necessary for using our multi-hop Bad Santa strategy.

We have shown that there are at least $1 + r + r^2$ node-disjoint paths from $N(a, b)$ to node p , all lying in in a single neighborhood $N(a, b + r + 1)$. Furthermore, we have shown that any correct node $p(x, y)$ can hear all one-hop and two-hop messages by round $(k+2)(\lfloor \frac{y-r-1}{r} \rfloor + 1)$. Node p can sample these messages over $k+1$ rounds and, since the $O(\log^{(k)}(n) + k)$ Bad Santa strategy is used for selecting $S_{p,i}$, node p is guaranteed to receive a correct message. This completes the induction.

In terms of resource bounds, we can again consider the situations where p must deal with (either broadcasting or receiving) a message: (1) p receives

messages in order to commit, (2) p broadcasts it has committed, and (3) p facilitates a two-hop message. We consider each case. To address (1), by the Bad Santa protocol, p listens to $O(\log^{(k)} n + k)$ messages in expectation. To address (2), note that p broadcasts that it has committed $k + 2 = O(k)$ times. To address (3), we note that p belongs to many B'_q sets for different nodes q ; however, regardless of which B'_q set, p only ever has at most two sister nodes. Therefore, considering broadcast along the x and y coordinates, the number of sister nodes is $O(1)$; the number of broadcasts due to two-hop paths messages is thus $O(k)$. The same arguments regarding fingerprints as given in the proof of Theorem 9 apply here which concludes the proof. This leads each node being awake over $O(\log^{(k)} n + k)$ time slots in expectation in Stage 2. Over both stages, each node sends $O(n \log^2 |m| + k|m|)$ bits, and listens to an expected $O(n \log^2 |m| + (\log^{(k)} n)|m|)$ bits. \square

4.4.7 Unknown Start Time and Source(s)

In our previous protocols, *both the source of the message and the time the message was sent out needed to be pre-established*. Furthermore, our previous protocol allowed a savings on the fraction of required awake time only in Stage 2. The new protocol we present here assumes that every $2r + 1$ by $2r + 1$ square contains $t < \frac{n}{16+\epsilon}$ faults. Specifically, we require that no more than a $1/2 - \epsilon$ fraction of the nodes are faulty nodes in any $r/2$ by $r/2$ square. The benefits of this protocol are that it is 1) more energy efficient than using the protocol of [22]; 2) avoids the need to have the source and sending time pre-specified; and 3) reduces the awake time over the entire execution. Therefore, this algorithm is preferable when the circumstances of the fault model permit.

Let Q_i refer to the set of nodes in some $\frac{r}{2} \times \frac{r}{2}$ square in the grid. Our algorithm relies on correctly transmitting a message m from Q_{i-1} to Q_i where Q_{i-1} and Q_i are disjoint and neighboring squares i.e. the squares are neighbors abutting each other. Critical to our algorithm is an *assignment* of nodes in Q_{i-1} to nodes in Q_i . This assignment can be viewed as an undirected bipartite graph with the two disjoint sets of vertices being Q_{i-1} and Q_i and the assignment represented via edges. For $p \in Q_{i-1}$ and $q \in Q_i$, p listens to q and q listens to p *if and only if there is an edge between p and q in the bipartite graph*. This assignment is constructed such that all but a small fraction of correct nodes in Q_i receive a majority of correct messages from correct nodes

in Q_{i-1} (and vice versa). This allows correct nodes in Q_i to majority filter on the messages they receive and decide upon the correct message. Thus, a message can be transmitted securely from $\frac{r}{2} \times \frac{r}{2}$ square to $\frac{r}{2} \times \frac{r}{2}$ square. For a message m and for any square Q_i to which m is sent by the above protocol, let $G(Q_i, m)$ be the set of correct nodes in Q_i that receive m after majority filtering over the accepted messages as described above. A result in [135] establishes the following theorem which we state without proof:

Theorem 13. *For any pair of squares Q_{i-1} and Q_i , there is non-zero probability of an assignment between nodes in Q_{i-1} and nodes in Q_i with the following properties:*

- *the degree of each node is at most a constant C which is independent of r ,*
- *if $|G(Q_{i-1}, m)| \geq (1/2 + \epsilon/2)|Q_{i-1}|$, then $|G(Q_i, m)| \geq (1/2 + \epsilon/2)|Q_i|$.*

We will refer to an assignment with the two properties stated in Theorem 13 as a *robust assignment*. Although a method of assignment is not specified, Theorem 13 guarantees that a robust assignment must exist. We now consider the problem of find such an assignment.

Corollary 2. *A robust assignment between squares Q_{i-1} and Q_i can be found in time that is exponential in r^2 .*

Proof. Consider any assignment between squares Q_{i-1} and Q_i as a bipartite graph G as described above. Both Q_{i-1} and Q_i have constant size $d = \frac{r^2}{4}$ so the number of possible bipartite graphs is at most $2^d \times 2^d = 2^{2d}$. Note that this is an upper bound on the number of different ways in which the faulty nodes can be placed in G . We know by Theorem 13 that there is non-zero probability that edges between Q_{i-1} and Q_i satisfy the property that a $(3/4 - \epsilon)$ -fraction of the nodes in Q_i have a majority of correct neighbors in Q_{i-1} . Consider each of the at most 2^{2d} possible configurations of faulty nodes. For each such configuration, check whether all correct nodes in Q_i have at least a $(3/4 - \epsilon)$ fraction of correct neighbors in Q_{i-1} . By Theorem 5, such a configuration must exist and can be found by this exhaustive search which requires examining at most $2^{2d} = 4^{(r^2/4)}$ graphs. \square

Therefore, for $r = \Theta(1)$, following the above procedure in Corollary 2 yields a robust assignment in constant time. Alternatively, a random regular graph will induce a desired assignment with probability at least $1 - 1/r^c$ for some constant $c > 0$. Recall that each node is assigned to C neighbors where C is independent of r . Therefore, for a sufficiently large value $r = \Theta(1)$, nodes are listening to a small fraction of a square. Finally, we note that it is sufficient to find one robust assignment and use it for all pairs Q_{i-1} and Q_i .

Protocol Using Robust Assignment

We now describe and argue the correctness of a simple algorithm for reliable broadcast which we call ALG. ALG operates in stages of $\eta = r^2/4$ rounds. We again assume the presence of a global broadcast schedule so that message collisions do not occur. Over all rounds, each node that has committed to a message will broadcast at its scheduled slot. At every η^{th} round, a node enters into the listening state for one full round. That is, during this η^{th} round, all nodes are listening to all nodes in its $\frac{r}{2} \times \frac{r}{2}$ square *throughout the round*. At the end of this round, if a node p has received an identical message from a majority of nodes in its $(2r + 1) \times (2r + 1)$ square, p commits to this message.

For all other $\eta - 1$ rounds in a stage, a node sends and listens as dictated by a robust assignment and the broadcast schedule. That is, a node p listens to node q 1) if and only if p and q are assigned to each other under the robust assignment; and 2) when q is scheduled to broadcast. A robust assignment can be found as stated in Corollary 2 prior to deploying the sensor network and this assignment can be preprogrammed into the nodes and used for all pairs of squares. Any node p may act as a source node. In this case, the source node will broadcast its message to its $r/2 \times r/2$ square in its time slot in an η^{th} round when all nodes are awake; the message should include a declaration that p is acting as a source. As in [21, 22, 24, 90], every node in the source's square commits to m and proceeds to broadcast m during their respective scheduled time slots. From this point, the message is propagated from $\frac{r}{2} \times \frac{r}{2}$ square to $\frac{r}{2} \times \frac{r}{2}$ square by sending and listening according to the robust assignment in a deterministic fashion: a square sends to the squares above and below and to the left and the right, in that order; Figure 4.6 illustrates this. Communication from one square to an adjacent square can be accomplished with a single round used per direction. Note that if η is

not divisible by 4, then we simply interrupt on the special η^{th} round, and continue with the next direction afterwards. Therefore, a correct node will know this order and listen to its adjacent squares using the corresponding robust assignment in accordance with the broadcast schedule. As before, we may assume that the partitioning of the network into squares and the ordering and synchronization issues are dealt with through the nodes' internal programming; these details are outside the scope of this work. The exact propagation of a message depends on the robust assignment used and the behaviour of the faulty nodes; however, we can show correctness for the task of reliable broadcast.

By Theorem 13, at least a $(1/2 + \epsilon/2)$ fraction of correct nodes in every square will eventually receive identical messages from the majority of nodes to which it has been assigned. At this point, such a correct node can commit to a message and begin broadcasting, again according to its robust assignment and scheduled time slots. Finally, we address the remaining fraction of correct nodes in a $r/2 \times r/2$ square that may not be able to commit to a message. Recall that at every η^{th} time slot, all nodes are listening for the entire round. Assuming that a $(1/2 + \epsilon/2)$ -fraction of the correct nodes in the square have committed to the correct message, this allows the remaining fraction of correct nodes in a square to majority filter on incoming messages during this round and commit to the correct message.

In terms of costs, note that each node is always listening to at most C time slots in each of $\eta - 1$ rounds and listening to $r^2/4$ time slots in the η^{th} round; a total cost of $C(\eta - 1) + r^2/4$ over η rounds. Therefore, each node sends $O(|m|)$ bits per round and has an amortized cost of $O(C)$ time slots per round and an amortized cost of $O(C|m|)$ bits per round. Since C is a constant, this establishes our claims in Theorem 5.

4.4.8 Practical Considerations

We finish off this section by remarking on more practical considerations regarding our protocols. To start, we note that the grid model that we have adopted for applying our Bad Santa protocols is fairly flexible. Empty locations in the grid may correspond to failed nodes or simply the absence of a device altogether. The work in [22] generalizes results on reliable broadcast on the grid to arbitrary graphs where the problem is defined in terms of

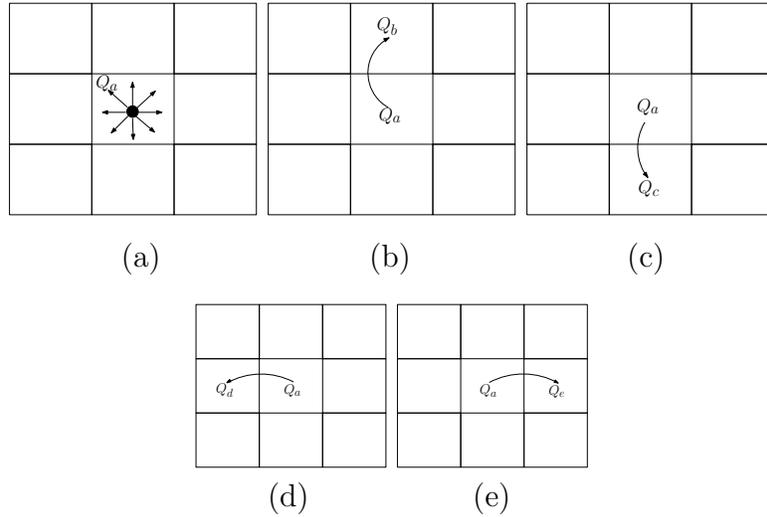


Figure 4.6: An illustration of the reliable broadcast protocol of Section 4.4.7. (a) Some node in Q_a acts as a source node to send a message to its square. Transmission from Q_a to (b) Q_b above, (c) Q_c below, (d) Q_d to the left and (e) Q_e to the right, using a robust assignment.

connectivity; our results easily generalize to such a setting and we refer the reader to [22] for more details. We also briefly mention that certain classes of random graphs may be mapped to the grid model; the details depend on the type of random graph utilized. For example, if nodes are placed uniformly at random in the two-dimensional plane, we can partition the plane into a grid and then map nodes to their nearest intersection point to achieve the grid model. In general, so long as the number of faults in a neighborhood does not exceed $t < (r/2)(2r + 1)$, this mapping will work. We are simply sketching this idea for the interested reader; clearly, the details of how many nodes need to be dropped to guarantee at most t faults in any neighborhood (with sufficient probability) and how the broadcast radius should be defined are details that we leave to future work. We refer the interested reader to work in [23] which deals with issues of probabilistic failures in the grid model and a random network model. Next, we offer some discussion on the aspects of the storage and processing overhead incurred by our algorithms, as well as some exploration of the utility of our protocols in terms of bit complexity savings.

Storage and Processing Overhead

Recall that devices in the sensor network are considered to be resource constrained. Here, we briefly discuss the costs associated with our algorithms in terms of processing and storage overhead, and we argue that these costs are reasonable. Note that these are costs that must be paid, to an even larger extent, in the original protocols of [21,22]. Furthermore, we argue that these costs are negligible in comparison to the cost of sending/receiving; hence, our algorithms do indeed achieve a energy savings.

In terms of storing data, consider our protocols where the send time and location of the source is known. Each node must store information on the current time slot, the slot when it can broadcast, its location relative to the source, its set of neighbors in the broadcast region, and information on the type of Bad Santa protocol being used (i.e. number of streams, the current stream, which time slots it is listening to); all of these can be stored with a small amount of overhead. The protocol for the case where we have Byzantine faults also requires storage of fingerprints, which are small compared to an actual message, and a hash function. The use of hash functions for such resource-constrained environments has been tested in [170] and in [95] (on the MICA series); it appears the storage costs are no obstacle. Therefore, the main storage overhead in our algorithms appears to result from messages. The length of these messages is likely application dependent and memory sizes can differ with the device in question. In [36], the MICA2 devices are stated to have 4 KB of memory. However, we note that current memory sizes on these sensor network devices can be sizable. For instance, in [171], the authors report that a flash-memory of 32 KB and the ability to add an additional storage capacity (up to 1 GB) for the devices studied. Therefore, memory size can be chosen for the application and corresponding message sizes in question; regardless, we do not anticipate that our algorithms incur an unreasonable amount of storage overhead over what is needed for storing messages.

The main processing cost of our algorithms differs per case. For the Byzantine fault-tolerant algorithm of Section 4.4.5, the main cost is likely due to the use of the hash function. Recall that this operation must be done fairly frequently in order for a node to commit to the correct message. While our algorithm employs a hash function, we note that public key cryptography (PKC) is not required by any of our algorithms. This distinction

is important since PKC has generally been considered to be expensive for energy constrained nodes due to the need for sending, receiving, and storing public keys and executing encrypt/decrypt operations [123]. More sophisticated techniques are now available which require less energy; however, the costs are still quite high. For instance, in [123], measurements using the MICA2DOT unit demonstrate a cost of 2302.70 mWs (milliwatt seconds) and 53.70 mWs for 2048-bit RSA signature generation and verification, respectively. Elliptic Curve Cryptography (ECC) is a popular alternative to RSA since it has smaller key sizes. For 224-bit ECC, the same authors measure costs with the MICA2DOT unit at 61.54 mWs and 121.98 mWs for signature generation and verification, respectively. Both RSA-2048 and ECC-224 are recommended by RSA Security as the new standard in order to protect data past the year 2010 [155]. These costs should be compared to the cost of broadcast on the Lucent IEEE 802.11 2Mbps WaveLAN PC Card which is measured at 266 mWs. Therefore, it is not clear that an algorithm could claim to save significant energy by employing public-key cryptographic schemes.

On the other hand, hash functions for energy-constrained environments have been considered in the literature and it appears the processing costs are reasonable [170]. In particular, the SHA-1 hash function can be applied with very little energy consumption; again with the MICA2DOT unit, the cost is measured to be 5.9 μ Ws/byte is measured in [155]. Therefore, hashing a 1 KB message would incur 5.9 mWs; notably this is far less than the cost of sending or receiving.

For the algorithm of Section 4.4.7, the most significant processing costs would appear to arise from the need to majority filter on all incoming messages; however, such a comparison operation is certainly feasible in sensor network devices. Finally, for the fail-stop case, the algorithm of Section 4.4.5 does not need to apply a hash function to messages and we do not anticipate significant processing costs here. In some cases, additional processing overhead will come from comparing hashes and accessing a random number generator; however, we anticipate that these additional processing overheads will be small in comparison to the cost of storing and processing messages.

Saving on Bit Complexity

Recall that our Byzantine fault-tolerant reliable broadcast protocol of Section 4.4.5 achieves asymptotically lower bit complexity through the use of hashing. However, there is the question of when such savings would be seen in practice. Packet sizes are discrete, and in many cases, the hash of a message may require the same number of packets as sending the message itself. If messages are small, then the bit complexity savings achieved by our protocol will be consequently smaller. However, we note that if messages are sizable then there is a benefit to the hashing technique.

In the face of large amounts of data collection and querying, data aggregation techniques have been proposed to reduce the overall communication costs since processing is generally less costly than sending data (see [59, 160] for more on this). Despite these techniques, there are applications for wireless networks that require transmission of large amounts of data even after processing. For instance, surveillance applications that require sending significant amounts of data have been proposed involving image and video data [40]. Therefore, there are indeed applications where large messages might be transmitted and we anticipate more such situations will arise in the future. Under such scenarios, we would expect our algorithms to save substantially on bit complexity.

When considering large messages, there is also the issue of slot size to consider. Modifying time division multiple access (TDMA) has been considered (see [74]) and it is possible that similar proposals could be used to allow large messages to be sent within a single time slot without underutilizing bandwidth. Alternatively, time slots could be reset by the dealer in order to accommodate large future transmissions; the details of this would likely be application specific and we leave this as a topic for future work.

Chapter 5

Message-Efficient Robust Communication in Distributed Hash Tables

In this chapter, we examine a communication setting that involves a set of multiple senders \mathcal{S} and a set of multiple receivers \mathcal{R} . As we will see, this setting arises in the peer-to-peer (P2P) paradigm which is a popular approach to providing large-scale decentralized services. However, the lack of admission control in many such systems makes them vulnerable to malicious interference [142, 153]. This is a practical concern since large-scale P2P systems are in existence today such as the Azureus DHT [49] and the KAD DHT [146], each of which see more than one million users per day. In addition to file sharing, there are proposals for using P2P systems to protect archived data [58], mitigate the impact of computer worms [9] and re-implement the Domain Name System [116]; such applications would likely benefit from increased security.

As mentioned in Chapter 1, there are a number of results on P2P systems that can provably tolerate Byzantine faults [12–14, 53, 75, 79, 112, 135]. To date, the majority of results pertain to distributed hash tables (DHTs). A common technique in DHTs that tolerate adversarial faults is the use of *quorums* which are sets of peers such that a minority of the members suffer adversarial faults. A quorum replaces an individual peer as the atomic unit. Adversarial behaviour can be overcome by majority action allowing for communication between correct peers; we call this *robust communication*. Since

critical operations such as data queries are performed in concert by members of a quorum, robust communication must be efficient and this is the focus of our work.

Before presenting our results, we lay the groundwork for the ideas in this chapter. First, we elaborate on the relevance of Byzantine fault tolerance in peer-to-peer systems in practice. Second, there have been a number of advances towards achieving such fault tolerance using quorums, and we present an overview of the main ideas present in the literature.

5.1 Quorums and Robustness

A popular approach to dealing with Byzantine attacks in DHTs is to use quorums [13, 53, 75, 112]. To reiterate, quorums are sets of peers with the property that a majority of the peers in a quorum have not suffered Byzantine faults. Typically, each quorum consists of $\Theta(\log n)$ peers where n is the number of peers in the network. Using quorums, it is possible to overcome adversarial behaviour by treating each quorum as the atomic unit instead of individual peers. If the Byzantine peers attempt to deviate from protocol, this errant behaviour can be overwhelmed through majority action. For instance, content may be stored in a distributed and redundant fashion over members of a quorum such that the content cannot be polluted by a single host peer. Poisoning attacks can be mitigated by having peers belonging to the same quorum validate content before it is advertised. Furthermore, a useful property of quorums is that those peers who violate protocol can be ejected from their quorums which effectively removes them from the system.

Several protocols using quorums have been proposed; however, there is a common theme in the way such quorums are utilized. A message m originating from a peer p traverses a sequence of quorums Q_1, Q_2, \dots, Q_ℓ until a destination peer is reached. A typical example is a query for content where the destination is a peer q holding a data item. Initially p notifies its own quorum Q_1 that it wishes to transmit m . Each peer in Q_1 forwards m to all peers in Q_2 . A peer in Q_2 determines the correct message by majority filtering on all incoming messages and, in turn, sends to all peers in the next quorum. This forwarding process continues until the quorum Q_ℓ holding p is reached. Assuming a majority of correct peers in each quorum, transmission of m is guaranteed. Unfortunately, this simple protocol is costly. If all

quorums have size s and the path length is ℓ , then the message complexity is $\ell \cdot s^2$. Typically, $s = \Theta(\log n)$ and, as in Chord [148], $\ell = O(\log n)$ which gives a $O(\log^3 n)$ message complexity which is likely prohibitively expensive for practical values of n .

Note that this approach can fail if the adversary obtains a majority of corrupted peers in any quorum. This can happen if an adaptive adversary has its peers join and leave the network until a favourable placement is attained. For instance, the adversary may target a quorum in the following manner. The adversary adds a corrupted peer p into the system to see where it is placed. If p lands within the target quorum, the adversary keeps the peer active in the system; otherwise, p leaves. Over a number of join-leave events the adversary may accumulate a large number of peers in the target quorum, eventually obtaining a majority. Remedies to this challenging *adaptive Byzantine adversary* have been proposed and we discuss this further in Section 5.2

To date, the previous best communication complexity for using quorums was given by Saia and Young [135] who give a randomized protocol which achieves $O(\log^2 n)$ messages in expectation over a path of length $O(\log n)$. While communication between two quorums incurs an expected constant number of messages, the analysis in [135] yields a prohibitively large constant. Furthermore, with probability $1 - o(1)$ some peers will incur $\omega(1)$ message complexity (see Section 5.6). The protocol also employs a link architecture between peers requiring the use of a Byzantine agreement protocol. Finally, maintenance and asynchronicity issues remain unresolved.

Therefore, while results exist on the feasibility of robust communication, work on the practicalities has lagged behind. This dearth presents an impediment to the deployment of such systems and we seek to address this outstanding problem.

5.2 Related Work

In this section, we expand on the introductory summarization of Byzantine fault tolerance that was provided in Chapter 2. We begin by summarizing a number of related applied results for achieving Byzantine fault tolerance. We then give an overview of a number of theoretical results.

State machine replication (SMR) is a standard method for implementing highly fault-tolerant services [138]. Services are replicated over multiple servers providing a high-integrity distributed system. While P2P systems do not align perfectly with the SMR paradigm [138], the literature on Byzantine fault-tolerant replication is relevant to our work. Early work by Reiter [128] gave protocols for Byzantine agreement and atomic broadcast. Our first protocol shares some common features with the multicast protocol of [128], yet we differ significantly since in the P2P domain we must contend with issues of scalability, churn, and spurious requests aimed at consuming resources. More recently, Castro and Liskov [32] demonstrated efficient Byzantine fault-tolerant SMR; however, this seems unsuitable for a P2P setting due to scaling issues. Several other Byzantine fault-tolerant systems exist such as SINTRA [30], FARSITE [2], the Query/Update protocol [1] and the HQ system [38]; however, these protocols have not been shown to scale to the P2P domain. More recent work on the FARSITE system allows for faulty machines to be identified [48]. However, empirical work and details, in particular details on secure information retrieval and message passing, are not provided in these works.

Two *implemented* large-scale Byzantine fault-tolerant storage architectures are OceanStore [92] and Rosebud [132]. The latter scales up to tens of thousands of nodes and handles changing membership. However, with only a single Byzantine node per replication group, Rosebud incurs significant overhead. In contrast, our protocols perform efficiently with 10% Byzantine peers. Rosebud relies on a *configuration service* (CS) which tracks system membership, ejects faulty nodes, and handles new nodes. The CS, implemented over a set of nodes, introduces a potential bottleneck and a possible point of attack; similarly, a “primary tier” of replicas is used in OceanStore. In contrast, our protocol is completely decentralized and no special set of nodes is required.

Both Rodrigues, Kouznetsov and Bhattacharjee [131] and Rodrigues, Liskov and Shrira [133] give *proposals* for applying the SMR approach on a large scale; the latter describes a P2P system. However, both works rely on a CS and neither provides empirical results or discusses the details of secure data retrieval and message passing. Wang *et al.* [157] design and implement a routing scheme that tolerates Byzantine faults and yields good performance. However, they require both a certificate authority (CA) and a special set of nodes, called a neighborhood authority, similar to a CS.

In summary, work on practical Byzantine fault-tolerance has focused on ensuring consistency and availability of replicas in the SMR paradigm; as noted in [131], this does not necessarily characterize the P2P domain. In many cases, the reliance on a special set of nodes introduces a potential bottleneck to the system; in contrast, our use of DKG and threshold cryptography removes this bottleneck. Finally, both empirically validated systems and proposals for larger-scale applications are either not aimed at, or do not directly address, the issue of robust communication in a P2P environment.

There are several theoretical results on Byzantine fault-tolerant DHTs [13, 53, 75, 112]. These results make use of *quorums*, which are sets of $\Theta(\log n)$ peers such that a majority of the peers in a quorum are correct. Awerbuch and Scheideler show how to tolerate a powerful adaptive adversary who attempts to gain a majority of Byzantine peers in a quorum [12–14]. Saia and Young [135] demonstrate more efficient robust communication but, as discussed earlier, several issues remain unresolved.

Castro *et al.* [31], Halo [82], and Salsa [111] handle Byzantine faults by routing along multiple diverse routes. The proposal in [31] requires a CA whereas we do not rely on any trusted third party. In both [82] and [111], the guarantees are unclear against an adversary who owns a large IP-address space or targets identifiers over time as described in [13]. Such an adaptive adversary could potentially compromise the “knuckle” nodes in [82] or the global contacts used in [111]; in contrast, defenses against an adaptive adversary are known for quorum-based protocols [12–14].

There are several other works relating to issues of security in P2P networks. The ShadowWalker system [105] addresses the issue of anonymity and routes securely using the notion of multiple “shadows” which are similar to a quorum; however, our protocols differ significantly. The Brahms system [28] allows for uniform sampling of peers despite a Byzantine adversary. The Fireflies architecture [79] allows each peer to remain informed of live members in the system despite Byzantine attacks; however, its applicability likely extends only to single-hop overlays such as in work by Gupta *et al.* [70] and secure routing in multi-hop networks is not treated.

5.3 Our Contributions

We improve over all previously known results involving communication between quorums [13, 53, 112, 135]. We summarize our main results below:

Theorem 14. *In the computational setting, for an adversary that controls up to an $\epsilon < 1/3$ -fraction of any quorum of size at most s , there are two protocols for achieving robust communication of a message m to a set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ . Our Robust Communication Protocol I (RCP-I) has the following properties:*

- *The total message complexity (number of messages sent and received) and the message complexity of the sending peer are each at most $2 \cdot s + 4 \cdot s \cdot (\ell - 2) + |D|$.*
- *The message complexity of every non-sending peer along the lookup path is at most 4.*
- *The latency (number of roundtrip communication rounds) is at most $2 \cdot (\ell - 2) + 2$.*

For our Robust Communication Protocol II (RCP-II):

- *The expected total message complexity and the expected message complexity of the sending peer are each at most $2 \cdot s + \frac{(\ell-2)}{(1-\epsilon) \cdot c} + (\ell - 2) + |D|$.*
- *The expected message complexity of a non-sending peer on the lookup path is at most $\frac{2}{(1-\epsilon) \cdot c \cdot s}$.*
- *The expected latency is at most $\frac{(\ell-2)}{(1-\epsilon) \cdot c} + 2$.*

Here, the constant $c > 0$ is the probability that the response time of a correct peer is at most Δ (see Section 5.6.1 for more details).

Using the Chord-based construction of [53], the message complexity of RCP-I is $O(\log^2 n)$ and for RCP-II it is $O(\log n)$ in expectation. We tolerate a large fraction of adversarial peers; strictly less than a 1/3-fraction compared to the roughly 1/4-fraction in [135]. Our use of a distributed key generation (DKG) scheme allows for security *without* a trusted party or costly updating

of public/private keys outside of each quorum. This obviates the need for a trusted third party. To the best of our knowledge, this is the first use of DKG in a Byzantine-tolerant P2P setting.

Finally, we provide microbenchmark results involving two quorums using PlanetLab. Our experimentation demonstrates that our protocols perform well under significant levels of churn and faulty behaviour. In particular, for a 10^5 -node system with $\ell = 20$, our results imply RCP-I and RCP-II complete in under 4 seconds and 5 seconds, respectively.

5.4 The Network Model

Each peer p is assumed to have a unique identifier, p_{ID} , and a network address, p_{addr} . Byzantine peers are also referred to as *faulty* or *adversarial*; all other peers are called *correct*. A fraction of the correct peers may crash due to a system failure or leave gracefully. We model such peers as having *crashed*.

We adopt an asynchronous communication model with unbounded message delivery time. However, for liveness in DKG and in our second protocol, we use a *weak synchrony* assumption by Castro and Liskov [33]. That is, let $\text{delay}(t)$ be the time between the moment t when a message is first sent and the moment when it is received, and assume the sender keeps retransmitting the message until it is received. We assume that $\text{delay}(t)$ does not grow faster than t indefinitely which, assuming that faults in the network are eventually repaired, seems to be valid in practice.

Peers p and q are said to communicate directly if each has the other in its routing table. The target of m is a set of peers D within a single quorum; m may be a data item request and D may consist of a single peer or multiple peers depending on how data is stored. Alternatively, m might signify a update operation (such as a peer leaving the network), in which case D could consist of multiple peers that need to be informed.

5.4.1 The Quorum Topology

There are several different approaches to how quorums are created and maintained [13, 112, 135]; we refer the reader to [53] for a detailed explanation.

Despite these different approaches, we may view the setup of quorums as a graph where nodes correspond to quorums and edges correspond to communication capability between quorums; we refer to this as the *quorum topology*. Figure 5.1 illustrates how quorums can be linked in a DHT such as Chord. We assume the following four simple invariants are true:

1. *Goodness*: each quorum has size $\Theta(s)$ for $s = \Omega(\log n)$ and possesses at most an ϵ -fraction of Byzantine peers for $\epsilon < 1/3$.
2. *Membership*: every peer belongs to at least one quorum.
3. *Intra-Quorum Communication*: every peer can communicate directly to all other members of its quorums.
4. *Inter-Quorum Communication*: if Q_i and Q_j share an edge in the quorum topology, then $p \in Q_i$ may communicate directly with any member of Q_j and vice versa.

These four invariants are standard in the sense that previous work on quorums in DHTs ensure they hold with probability nearly equal to 1. For simplicity, we assume that these invariants hold with probability 1. However, many previous results make guarantees with high probability: $1 - 1/n^k$ for any desired constant $k > 0$. In such cases, our results also hold with high probability. For example, results for maintaining the goodness invariant in DHTs are known [12–14]. For the membership invariant, there exist quorum topologies where a peer may belong to several different quorums simultaneously [53, 112]. Finally, to the best of our knowledge, no implementation of a quorum topology exists; this represents another gap between theory and practice. A number of challenges remain in bridging this gap and such an endeavor is outside the scope of this current work. However, the literature suggests that, with the proper deployment, maintaining these invariants in real-world DHTs is plausible.

Consider a peer $p \in Q_i$. Peer p maintains a membership list of peers which p believes belong to Q_i . We note that peers will likely have different views of the network and hence membership lists for Q_i will likely differ for two peers; however, such issues can be overcome (see [53]) and this does not impact the use of eviction as a penalty for misbehaving peers. Throughout, we will refer to the eviction of a peer p from a quorum. By this we mean that

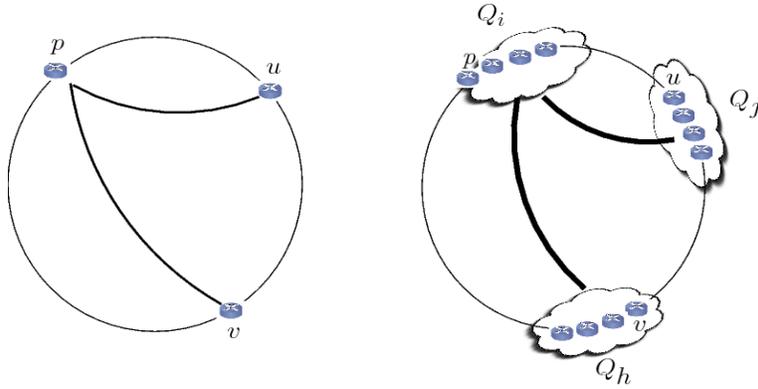


Figure 5.1: (Left) Three peers on a DHT ring where p links to u and v . (Right) An example of a quorum topology in a DHT ring where $p \in Q_i$, $u \in Q_j$ and $v \in Q_h$. Thick lines signify inter-quorum links.

a sufficient number of peers in Q_i remove p from their respective membership lists for Q_i and stop responding to requests by p . An evicted peer cannot perform any actions in the network and must rejoin the network. We assume that rejoining is an expensive operation as discussed in Section 5.4.4.

5.4.2 Threshold Cryptography and Distributed Key Generation

One way in which we achieve efficient communication is the use of threshold signatures to authenticate the communication between quorums. The idea behind an (η, t) -threshold scheme is to distribute a secret key among η parties in order to remove any single point of failure. Any subset of more than t parties can jointly reconstruct the secret key in the presence of a Byzantine adversary who controls up to t parties.

Threshold Signatures: In an (η, t) -threshold signature scheme, a signing (private) key k is distributed among η parties by a trusted dealer using a verifiable secret sharing protocol [51] or by a completely distributed approach using a DKG protocol [118]. The distribution algorithm generates (1) a private key shares k_i for each party, (2) a verification (public) key K , and (3) the associated public key shares \hat{K} . Any subset of $t + 1$ or more parties may

sign a message m by using their private key shares to generate the signature shares σ_i . Any party can combine these signature shares to form a message-signature pair $S = (m, \sigma) = [m]_k$ that can be verified using the public key K ; however, this does not reveal k . We refer to a message-signature pair S as a signature. It is also possible to verify σ_i using the public key shares \widehat{K} . We assume that no computationally bounded adversary that corrupts up to t parties can forge a signature $S' = (m', \sigma')$ for a message m' . Further, malicious behaviour by up to t parties cannot prevent generation of a signature.

The threshold version [25] of the Boneh-Lynn-Shacham (BLS) signature scheme [27] is particularly well suited to our setting since it does not require a trusted dealer for key generation, and significant interaction between parties is avoided. Furthermore, both the signature size and generation algorithm are more efficient than other practical threshold signature schemes such as the DSS [60] and RSA signature schemes [141]. Therefore, to authenticate the communication between the quorums, we use the threshold BLS signature scheme.

Distributed Key Generation (DKG): Since we cannot rely on a trusted party in the P2P paradigm, we use a DKG scheme to generate the distributed private key. An (η, t) -DKG protocol allows a set of η nodes to construct a shared secret key k such that its shares k_i are distributed over the nodes and no set of fewer than t nodes may reconstruct the secret; no trusted dealer is required. There is also an associated public key K and a set of public key shares \widehat{K} for verification.

The protocol in [85] is the first DKG for an asynchronous setting; therefore, it is uniquely suitable for deployment in a P2P network. Along with a Byzantine adversary, this protocol also tolerates crash failures. For a quorum of size $s = \eta$, with t Byzantine nodes and f correct nodes that can crash, the DKG protocol requires that $s \geq 3t + 2f + 1$. In our case, this *security threshold* holds due to the goodness invariant in Section 5.4.1. The DKG protocol allows for system dynamics without changing the system public key K and this can be done efficiently by batching; details are given in Section 5.7.2. Notably, the message complexity of a batch of peers (say set P) all joining and/or all leaving the quorum is the same as for a single peer joining/leaving the quorum, while the bit complexity increases only linearly with $|P|$ (see [85]); for efficiency, we batch such operations during our analysis in Section 5.7.2.

5.4.3 Spamming Attacks

A critical concern is that the adversary may launch spurious communications aimed at consuming resources; we refer to such behaviour as *spamming*. For example, a malicious peer may initiate a number of data retrieval requests [142, 153]. Here the situation is more dire since the impact of such attacks is multiplied by the group action in a quorum-based system.

Ultimately, there is no perfect defence against an adversary with the resources to initiate massive spamming attacks or denial-of-service (DoS) attacks, and this is not our focus. In such circumstances, spamming amounts to a denial-of-service (DoS) attack which forces correct peers to quit the system; unfortunately, there appears to be no adequate remedy. On the other hand, an extended cuckoo rule exists for maintaining the goodness invariants against limited DoS attacks [14] and this result is compatible with our proposal. Regardless, handling massive spamming or DoS attacks is a challenging problem that falls beyond the scope of this current work. Rather we show that our protocols do not afford the adversary an advantage in launching such attacks. Our goal is to prevent the adversary from forcing a peer to perform expensive operations with impunity. For any operation initiated by a spammer p , this can be accomplished by either (A) placing the bulk of the cost of executing said operation on p or (B) making the detection of spamming inexpensive. As we will show in Section 5.5, our protocol RCP-I in Section 5.5.1 employs principle (A) while our protocol RCP-II in Section 5.6.1 employs principle (B).

In addition to cryptographic techniques, we assume a *rule set* to reduce the impact of spamming attacks as introduced by Fiat *et al.* [53]. A rule set defines acceptable behaviour in a quorum; for example, the number of data lookup operations a peer may execute per duration of time, or tit-for-tat behaviour for uploads/downloads. Such rules are known to everyone within a quorum and can be implemented at the software level or simply agreed upon by quorum members. As discussed in Section 5.1, requests from a peer q who deviates from the rule set are ignored by the other members of its quorum, effectively removing q from the system.

5.4.4 Feasibility via Quorums, but Efficiency via Cryptography

We now make explicit a crucial point regarding the feasibility versus the efficiency of robust communication. As we have discussed, in the presence of Byzantine peers, no single peer can be trusted and quorums are employed to overcome this trust deficit through majority action. Using the simple protocol outlined in Section 5.1, the transmission of a message is guaranteed. *Therefore, quorums allow for robust communication without the need for cryptographic techniques.* However, as we now discuss, cryptographic techniques are important to achieving efficient robust communication.

A Problem of Spamming: Note that spamming attacks can pose a critical problem in a system that employs quorums. For example, a group of Byzantine peers may pretend to be a quorum and initiate requests. Therefore, simply obeying a request because it *appears* to come from a quorum does not prevent spamming. To investigate the implications of spamming, consider the case where peers act on any received request and call this the *passive scenario*. In the passive scenario, a Byzantine peer p can contact *any* quorum Q_i by colluding with other faulty peers to obtain necessary routing information. Members of Q_i act on any request coming from p . Even if it is possible to detect spurious requests at a global scale, each correct peer would be required to maintain $O(n)$ records to exclude faulty peers from the system.

Therefore, the passive scenario is undesirable since spamming allows the adversary to consume the resources of correct peers at little cost to itself. A standard fix is that a quorum responds only to requests that are “proven” to be legitimate. Yet, there is a cost to proving legitimacy; we explore this to motivate our protocols. First, we expand on the utility of a quorum topology in proving legitimacy. We then show how cryptographic techniques improve the efficiency of this task.

Legitimacy and the Utility of the Quorum Topology: We now contrast the passive scenario against another general scenario that we call the *prove-and-verify scenario* which assumes that proofs and verifications are required to initiate operations. We argue that the prove-and-verify scenario is superior to the passive scenario and discuss how the quorum topology is used in previous works to provide the framework for proving and verifying operations.

P2P systems often lack admission control and, if forced to leave the system, a Byzantine peer may simply rejoin the network with a new identity. In the worst case, perpetual and rapid rejoin operations result in a DoS attack. Therefore, we make the standard assumption that there is a cost for joining the network. For example, monetary costs are suggested in [31] and CAPTCHAs are suggested in [111]. Let τ denote the rate at which p can issue spurious requests before being forced to rejoin the system. In the passive scenario, a Byzantine peer p can contact *any* quorum Q_i by colluding with other faulty peers to obtain necessary routing information and so τ is large due to the abundance of potential targets.

In contrast, in the prove-and-verify system the members of Q_i must verify p 's proof before acting. Proof and verification may take different forms. For instance, constructions exist where two peers communicate only if their respective quorums are linked [53, 112]; that is, the quorum topology itself acts as proof. Verification occurs by having a quorum Q_i act on p 's request only if each peer in Q_i receives messages from a majority in Q_p . Here τ is greatly reduced. Furthermore, correct peers are not required to maintain records on misbehaving peers. However, while the prove-and-verify scenario is far more robust to spamming, there are shortcomings to this actual method of proof and verification and we discuss this next.

Efficiency in the Prove-and-Verify Scenario: We argue two things: (1) the form of proof discussed above is restrictive and (2) verification is expensive. First, the proof is restrictive since for Q_i and Q_j to communicate without sending through intermediary quorums, they must maintain links to one another; such maintenance is costly. Second, the verification process is expensive because when communication occurs from Q_i to Q_j , a correct peer $q \in Q_j$ must know to which peers in Q_i it must listen; this incurs more maintenance costs. These are two significant problems with existing schemes.

Cryptography allows us to improve asymptotically on the message complexity of verification. Under our protocols, each quorum has a public and private key established using DKG. Communication can occur between any two quorums that know and can verify each other's public key. Therefore, the form of proof is not as restricted by the quorum topology and we exploit this in RCP-II. Furthermore, verification is cheaper, using $O(s)$ messages in RCP-I or $O(1)$ expected messages in RCP-II. Of course, overhead is incurred by using cryptography. Message sizes increase by an additional number of bits dependent on the security guarantees, and keys shares, but *not* the key

itself, must be updated when membership changes. However, our experimental results in Section 5.7 show that this overhead is tolerable since the computation costs are significantly smaller than the network latency. Hence, cryptography provides a more efficient and flexible implementation of the prove-and-verify scenario.

5.5 Robust Communication Protocols

We propose two robust communication protocols: RCP-I and RCP-II. Here we outline a general scheme in Figure 5.2 that is later refined to give our two protocols. Consider a sending peer p who wishes to send a message m to peer p' . We assume m is associated with a key value which yields information necessary for distributed routing; that is, the next peer to which m should be forwarded is always known. Peer p notifies its quorum Q_1 that it is performing robust communication and receives $\text{PROOF}(Q_1)$. Peer p sends this to Q_2 as proof that p 's actions are legitimate; the form of this proof is discussed later. Depending on the scheme, one or more members of Q_2 examines the proof and, upon verifying it, sends to p : (1) routing information for Q_3 and (2) $\text{PROOF}(Q_2)$, that will convince Q_3 that p 's actions are legitimate. This continues iteratively until p contacts the quorum holding p' and m is delivered. We employ the following concepts:

Quorum Public/Private Keys: Each quorum Q_i is associated with a (distributed) public/private key pair (K_{Q_i}, k_{Q_i}) ; however, there are two crucial differences between how such a key pair is utilized here in comparison to traditional implementations. First, only those quorums linked to Q_i in the quorum topology, and not everyone in the network, need to know K_{Q_i} . Second, (K_{Q_i}, k_{Q_i}) is created using the DKG protocol and \widehat{K}_{Q_i} is the associated set of public key shares.

Individual Public/Private Key Shares: Each peer $p \in Q_i$ possesses a private key share $(k_{Q_i})_p$ of k_{Q_i} produced using DKG. Unlike the quorum public/private key pair of Q_i which must be known to all quorums to which Q_i is linked in the quorum topology, only the members of Q_i need to know the public key shares \widehat{K}_{Q_i} , which plays an important role in allowing members of Q_i to verify that the signature share sent to peer p is valid.

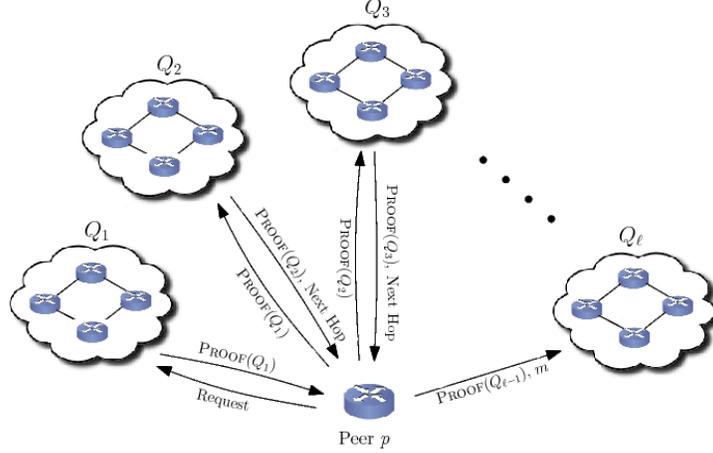


Figure 5.2: Our general robust communication scheme. At step $i = 1, \dots, \ell - 1$, peer p presents proof, $\text{PROOF}(Q_i)$, that quorum Q_i sanctions p 's action, and receives new proof from Q_{i+1} in addition to routing information for the next hop. At the final step ℓ , peer p sends $\text{PROOF}(Q_{\ell-1})$ and m .

5.5.1 Robust Communication Protocol I

We now illustrate RCP-I for a peer p who wishes to send a message m . The path m takes through quorums is denoted by Q_1, \dots, Q_ℓ . We assume that $p \in Q_1$ and the target of the message is a set of peers $D \subseteq Q_\ell$.

Overview: We outline RCP-I; the pseudocode is given in Figure 5.3. Initially, the correct peers of Q_1 must acquiesce to p 's request. Peer p begins by sending $[p_{\text{id}}|p_{\text{addr}}|\mathbf{key}|ts_1]$ to all peers in its quorum Q_1 . The value \mathbf{key} corresponds to the intended destination of m and ts_1 is a time stamp. The message m can also be sent, and its hash can be added inside the signature below; however, for simplicity, we assume m is sent only in the last step. Each correct peer $q \in Q_1$ then consults the rule set and sends its signature share to p if p is not in violation of the rule set to within some bound to compensate for clock drift. Peer p interpolates these signature shares to generate the signature: $S_1 \leftarrow [p_{\text{id}}|p_{\text{addr}}|\mathbf{key}|ts_1]_{k_{Q_1}}$.

In each intermediate step $i = 2, \dots, \ell - 1$, p sends its most recent signature S_{i-1} for p 's request and a new time stamp ts_i to each peer $q \in Q_i$ along the lookup path. Since Q_i is linked to Q_{i-1} in the quorum topology, each q knows the public key $K_{Q_{i-1}}$ to verify S_{i-1} . If S_{i-1} is verified and ts_i is

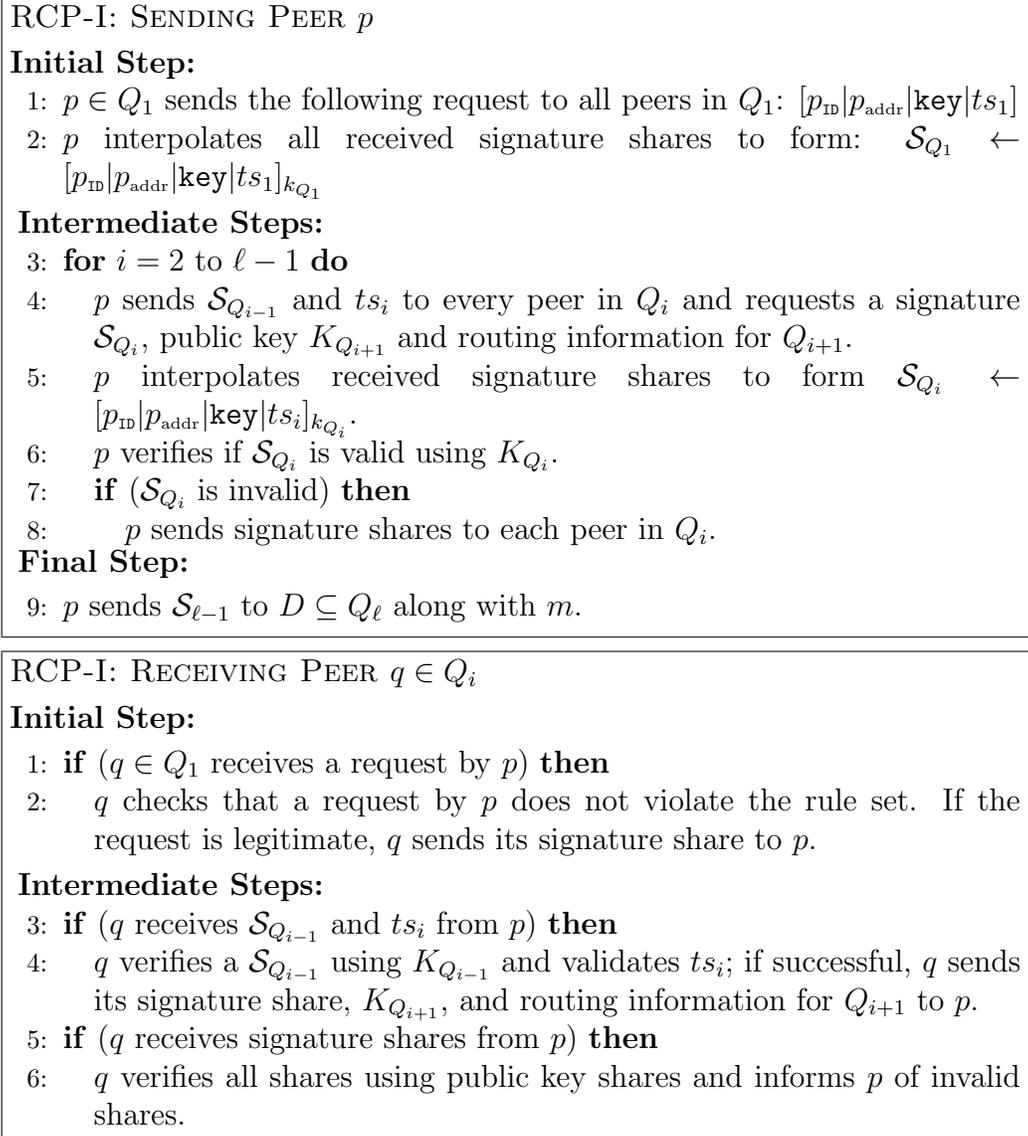


Figure 5.3: Pseudocode for RCP-I

valid, q sends back its signature share for the request message, $K_{Q_{i+1}}$ and the routing information. Peer p collects the shares to form S_i and majority filters on the routing information for Q_{i+1} . Finally, for Q_ℓ , p sends m along with $\mathcal{S}_{\ell-1}$ to peers in the set D .

Share Corruption Attack: Note the following attack: a set of Byzantine peers $B \subsetneq Q_i$ send invalid shares to p and, therefore, p will fail to construct S_i . We refer to this attack as the *share corruption attack*. Here, the individual public/private key shares play a crucial role. To obtain S_i , p sends the received shares to each peer in Q_i using one message per peer. For a share sent to p by a peer in Q_i , each correct peer in Q_i verifies the share using \widehat{K}_{Q_i} . All valid shares are then sent back to p who creates S_i . Note that the shares are not recomputed; hence, the adversary can only perform this attack once per step. Also, while peers in Q_i may identify the peers which p alleges sent an incorrect share, punitive action is limited, since p itself may be lying about these transactions.

Lemma 24. *RCP-I guarantees that m is transmitted to a target set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ with the following properties:*

- *Both the total message complexity and the message complexity of the sending peer is each at most $2 \cdot s + 4 \cdot s \cdot (\ell - 2) + |D|$.*
- *Each forwarding peer has message complexity at most 4 messages.*
- *The latency is at most $2 \cdot (\ell - 2) + 2$.*

Proof. First, we prove correctness. We show that if p is correct and has not violated the rule set, at each step i of the protocol p either (1) receives a valid signature and routing information for the next step or (2) terminates the protocol by delivering m to all members of D ; correctness follows directly. Our proof is by induction on i :

Base Case: Consider the initial step $i = 1$ where p communicates with the peers in its quorum $Q_p = Q_1$ about sending the message m . If p is correct and has not violated the rule set, upon receiving $[p_{\text{id}}|p_{\text{addr}}|\text{key}|ts_1]$ all correct peers will send their shares to p . Therefore, p is guaranteed to form \mathcal{S} by the goodness invariant. Peer p can then check whether S is valid and, if so, sets S to be S_1 . Otherwise, p must overcome the share corruption attack. Since p belongs to Q_1 , peer p knows the individual public key shares of each peer in Q_1 and can therefore detect which shares are invalid and construct S_1 . Finally, p already has the routing information for Q_2 ; therefore, the base case holds.

Inductive Hypothesis: Assume that up to step $i < \ell$, p has obtained the correct signatures and routing information.

Inductive Step: At step $i + 1$, peer p sends S_i to Q_{i+1} . By the inductive hypothesis, this signature is valid and p possesses the routing information for Q_{i+1} . If $i < \ell - 1$, and no corrupted share attack occurs, then p 's request for S_{i+1} and the routing information for Q_{i+2} will be satisfied due to the goodness invariant. Otherwise, p must overcome the corrupted share attack by sending all signed shares to all other peers in Q_{i+1} . Each correct peer in Q_{i+1} can detect and inform p which peers sent an invalid share. Due to the goodness invariant, peer p can majority filter on these responses to determine the invalid shares and then construct S_{i+1} . If $i = \ell - 1$, p possesses the routing information for Q_ℓ to deliver m to all members of $D \subseteq Q_i$ and the protocol terminates successfully. In either case, the induction holds.

We now analyze the costs of our protocol. In the first step, even in the event that a share corruption attack occurs, at most one round-trip round of communication occurs (between p and Q_1 since p holds \widehat{K}_{Q_1}). For steps $i = 2, \dots, \ell - 1$, if a share corruption attack occurs, at most two round-trip rounds of message exchange occur: (1) p sends to Q_i and Q_i sends back to p and (2) p transmits shares to Q_i who then send the correct shares back to p . Adding the last step, the latency is $2 \cdot (\ell - 2) + 2$. In terms of message complexity, in the first round, peer p must send a request to and receive a response from each peer in Q_1 ; this totals at most $2s$ messages. For steps $i = 2, \dots, \ell - 1$ peer p must both send a request to and receive a response from each peer in a quorum; if a corruption attack occurs, p must send another message to each peer in a quorum (with all signed shares collected together) and receive back a response. Therefore, this incurs at most $4 \cdot s$ messages. In the last step p sends to all members of D . Hence, the message complexity is at most $4 \cdot s \cdot (\ell - 2) + |D| + 2s$. For every other involved peer $q \notin D$, q 's message complexity is at most 4; clearly, peers in D receive one message. \square

Spamming Attacks: The sending peer p experiences more cost than other participating peers. In part, this is due to the iterative nature of the protocol; however, largely this is because p must send and receive $O(s)$ messages per step. In contrast, other participating peers need only send and receive a constant number of messages over the execution of the protocol.

Peer p may misbehave in other ways. For instance, p may repeatedly contact its quorum to initiate robust communication; however, eventually

all correct peers will ignore p . Similarly, using a correct signature, p may repeatedly ask q in another quorum for proof and/or routing information; however, time stamps limit such replay attacks. In conclusion, such actions cannot cause correct peers to perform expensive operations.

5.6 Determinism is an Improvement

We conclude our discussion of RCP-I by showing that, under the previous protocol of Saia and Young [135], some correct nodes will likely still incur non-constant message complexity; the determinism of RCP-I avoids such imbalances. To be fair, we note that in [135] the worst-case complexity is still relatively small for realistic values of n , but we include this analysis for completeness.

Theorem 15. *Consider the first lookup scheme in [135]. In each hop, with probability $1 - o(1)$, there exists a peer that experiences $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$ message complexity. Over $\Theta(\log n)$ hops, with probability $1 - o(1)$, $\Theta(\log n)$ peers each experience a message complexity of $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$.*

Proof. Consider a message being passed from quorum L to quorum R . The scheme of [135] works as follows. Let B denote a set of $\ln n$ bins. Each peer in R is mapped to C bins in B uniformly at random. Then each peer in L is mapped to a single bin in B uniformly at random. A peer in R listens only to those peers that get mapped to the same bin as it does. Our first goal is to show that, with probability $1 - o(1)$, there exists some bin to which $\Omega(\ln \ln n / \ln \ln \ln n)$ peers in R get mapped. We proceed by adapting the balls-and-bins argument of [106]. Assuming the Poisson distribution, let p_k be the probability that there exists at least one bin which has at least k peers. Then:

$$p_k \geq \frac{\left(\frac{C \ln n}{\ln n}\right)^k}{k! \cdot e^{\frac{C \ln n}{\ln n}}} = \frac{C^k}{k! \cdot e^C} \quad (5.1)$$

The probability that no bin has at least k balls is at most $(1 - p_k)^{\ln n} \leq e^{-p_k \cdot \ln n}$. We wish to now show that $\frac{1}{(\ln n)^2} > e^{-p_k \cdot \ln n}$ as this implies that, with probability $1 - o(1)$, there exists some bin to which $O\left(\frac{\ln \ln n}{\ln \ln \ln n}\right)$ peers in R get mapped.

Taking the logarithm of each side twice (and noting $e^{-\frac{C^k}{k! \cdot e^C} \cdot \ln n} \geq e^{-p_k \cdot \ln n}$ by Equation 5.1), this is equivalent to proving: $k \cdot \ln C + \ln \ln n > \ln 2 + \ln \ln \ln n + \ln(k!) + C$. Substituting $k = \frac{\ln \ln n}{\ln \ln \ln n}$, the left side of the inequality is at least: $\left(\frac{\ln \ln n}{\ln \ln \ln n}\right) \cdot \ln C + \ln \ln n$. For the right side, we use Stirling's approximation for $x! = \sqrt{2\pi x} \cdot (x/e)^x \cdot (1 + O(1/x)) \leq 2\sqrt{2\pi x} \cdot (x/e)^x$ for sufficiently large x . Then: $\ln(k!) \leq \ln(2\sqrt{2\pi k} \cdot (k/e)^k) \leq \ln(\sqrt{8\pi k}) + k \ln(k)$ for sufficiently large k . Then the right side of the inequality is at most $\ln 2 + \ln \ln \ln n + \ln\left(\sqrt{8 \cdot \pi \cdot \frac{\ln \ln n}{\ln \ln \ln n}}\right) + \ln \ln n + C$. Therefore, for sufficiently large n , the inequality $\frac{1}{(\ln n)^2} > e^{-p_k \cdot \ln n}$ holds. Now, by the results in [106] relating tail bounds of the Poisson and binomial distributions, the result holds for the binomial distribution up to a constant factor.

We will call a bin *overloaded* if it has $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$ peers mapped to it under the scheme in [135]. We now consider the number of peers that get mapped from L to an overloaded bin over $\ell = \Theta(\log n)$ steps of the message passing protocol in [135]. Let X_i be the indicator random variable that has value 1 if in step i of the protocol, at least one overloaded bin exists (which is $1 - o(1)$ by the above argument) and at least one peer in L is mapped to an overloaded bin; otherwise, X_i is zero. Then $\Pr[X_i = 0] \leq o(1) + (1 - 1/\ln n)^{C \ln n} \leq o(1) + e^{-C}$ for each step $i = 1, \dots, \ell$. Therefore, $\Pr[X_i = 1] \geq 1 - o(1) - e^{-C} = \Theta(1)$. Let $X = \sum_i^\ell X_i$, then by linearity of expectation $E[X] = \Theta(\log n)$. Since the X_i s are independent and i.i.d., by standard Chernoff bounds: $\Pr[X < (1 - \delta) \cdot E[X]] < e^{-\Theta(\log n)} = n^{-\Theta(1)} = o(1)$. Therefore, with probability at least $1 - o(1)$, over $\Theta(\log n)$ steps of the protocol, $\Theta(\log n)$ peers will have message complexity $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$. \square

5.6.1 Robust Communication Protocol II

RCP-II is a randomized algorithm yielding a small expected message complexity for both the sending peer and forwarding peers. In exchange, join and leave operations incur additional cost in comparison to RCP-I; we discuss this in Section 5.6.2.

RCP-II utilizes signed routing table information. As a concrete example, we assume a Chord-like DHT although other DHT designs can be accommodated. For a peer $u \in Q_i$, each entry of its routing table has the form $[Q_j, p_{\text{ID}}, p'_{\text{ID}}, K_{Q_j}, ts]$. Here $p \in Q_j$ and $p' \in Q_{j-1}$ where (1) Q_i links to Q_j

and Q_{j-1} in the quorum topology, (2) Q_{j-1} immediately precedes Q_j clockwise in the identifier space and (3) p and p' are respectively located clockwise of all other peers in Q_j and Q_{j-1} . K_{Q_j} is the quorum public key of Q_j , and ts is a time stamp for when this entry was created. Note that any point in the identifier space falls between unique points p_{ID} and p'_{ID} . Given this property, and that entries are signed by a quorum, any attempt by a malicious peer along the lookup path to return incorrect routing information can be detected. \mathcal{RT}_{Q_j} denotes the routing table information for all peers in Q_j . $[K_{Q_j}]_{k_{Q_i}}$ is the quorum public key of Q_j signed using the private quorum key of Q_i ; recall neighbors in the quorum topology know each others' public keys. $[\mathcal{RT}_{Q_j}]_{k_{Q_i}}$ is the routing information signed with the private key of Q_i ; entries of the routing table are signed separately. Routing table information is time stamped and re-signed periodically when DKG is executed.

Overview: We sketch RCP-II here. For simplicity, we temporarily assume that peers act correctly; our pseudocode in Figure 5.4 is complete for when peers fail to respond to requests by p . Initially, each correct peer in Q_1 receives $[p_{ID}|p_{addr}|\mathbf{key}|ts]$ from p . The time stamp ts is chosen by p and peers in Q_1 will acquiesce to the value if it agrees with the rule set to within some bound to compensate for clock drift. If the request does not violate the rule set, then the information is signed allowing p to form $M_1 = [p_{ID}|p_{addr}|\mathbf{key}|ts]_{k_{Q_1}}$.

In the second step of the protocol, p knows the membership of Q_2 and selects a peer $q_2 \in Q_2$ uniformly at random (u.a.r.) without replacement. Peer p then sends M_1 to q_2 . Assuming q_2 is correct, it verifies M_1 using K_{Q_1} and checks that the ts is valid; the duration for which a time stamp is valid would be specified by the rule set. Once verified, q_2 sends p the information $[K_{Q_1}]_{k_{Q_2}}$, $[\mathcal{RT}_{Q_3}]_{k_{Q_2}}$ and $[K_{Q_3}]_{k_{Q_2}}$. Peer p knows K_{Q_2} since Q_1 links to Q_2 and verifies $[K_{Q_1}]_{k_{Q_2}}$, $[\mathcal{RT}_{Q_3}]_{k_{Q_2}}$ and $[K_{Q_3}]_{k_{Q_2}}$, and checks that the time stamp on the routing information is valid. If so, p constructs $M_2 = [M_1|[K_{Q_1}]_{k_{Q_2}}]$. Here $[K_{Q_1}]_{k_{Q_2}}$ will allow some peer in Q_3 to verify K_{Q_1} and M_1 , while the signed verified K_{Q_3} will allow p to check the response from that peer in Q_3 .

This process repeats with minor changes for the remaining steps. Using \mathcal{RT}_{Q_3} from the previous step, p selects a peer q_3 randomly from Q_3 and sends M_2 . Since Q_3 is linked with Q_2 in the quorum topology, q_3 knows K_{Q_2} ,

which it uses to verify $[K_{Q_1}]_{k_{Q_2}}$; this allows q_3 to verify M_1 signed with k_{Q_1} . Peer q_3 then confirms that ts is valid and sends $[K_{Q_2}]_{k_{Q_3}}$, $[\mathcal{RT}_{Q_4}]_{k_{Q_3}}$ and $[K_{Q_4}]_{k_{Q_3}}$ to p . Peer p has a verified public key K_{Q_3} from the previous step and uses it to verify $[K_{Q_2}]_{k_{Q_3}}$, $[\mathcal{RT}_{Q_4}]_{k_{Q_3}}$, and $[K_{Q_4}]_{k_{Q_3}}$. Then p constructs $M_3 = [M_2|[K_{Q_2}]_{k_{Q_3}}] = [M_1|[K_{Q_1}]_{k_{Q_2}}|[K_{Q_2}]_{k_{Q_3}}]$. This process continues until m is delivered. Group membership and signed routing information is agreed upon through the use of the DKG protocol. Figure 5.4 gives the pseudocode for RCP-II. Every peer contacted by p verifies a chain of certificates, which can be converted into a single signature using the concept of aggregate signatures [26].

Peer p may choose a Byzantine peer that does not respond. In that case, after an appropriate time interval, p will select an additional peer in the quorum. Let X be a random variable denoting the time required for a correct peer to respond. We make a weak assumption that $Pr[X \leq \Delta] \geq c$ where Δ is any duration of time and $c > 0$ is any constant probability. This does not circumscribe a particular distribution for response times; *any* distribution suffices, including the Poisson, exponential, and gamma distributions previously used to characterize round trip time (RTT) over the Internet. In practice, peer p would set its own Δ by sampling the network using methods for estimating RTT [78]. Since at most a constant fraction of peers are Byzantine, taking the median from a sufficiently large sample will determine Δ and p will receive a response from any of the previously selected peers — *this is in accordance with the weak synchrony assumption* stated in Section 5.4.

Lemma 25. *RCP-II guarantees that m is transmitted to a target set of peers $D \subseteq Q_i$ for some quorum Q_i over a path of length ℓ with the following properties:*

- *Both the total message complexity and the message complexity of the sending peer is each at most $2 \cdot s + \frac{(\ell-2)}{(1-\epsilon) \cdot c} + (\ell - 2) + |D|$.*
- *Each forwarding peer has expected message complexity at most $\frac{2}{(1-\epsilon) \cdot c \cdot s}$.*
- *The expected latency is at most $\frac{(\ell-2)}{(1-\epsilon) \cdot c} + 2$.*

RCP-II: SENDING PEER p

Initial Step:

- 1: p sends the following to each peer $q \in Q_1$: $[p_{\text{ID}}|p_{\text{addr}}|\mathbf{key}|ts]$
- 2: p gathers all responses and constructs:

$$M_1 \leftarrow [p_{\text{ID}}|p_{\text{addr}}|\mathbf{key}|ts]_{k_{Q_1}}$$

Intermediate Steps:

- 3: **for** $i = 2$ to $\ell - 1$ **do**
- 4: **while** (p does not have M_i and has waited time Δ since previous selection) **do**
- 5: p sends M_{i-1} to $q \in Q_i$ selected u.a.r. without replacement.
- 6: **if** ($[K_{Q_{i-1}}]_{k_{Q_i}}$, $[\mathcal{RT}_{Q_{i+1}}]_{k_{Q_i}}$ and $[K_{Q_{i+1}}]_{k_{Q_i}}$ are received from any peer in Q_i previously selected) **then**
- 7: p uses K_{Q_i} to verify $K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$.
- 8: **if** ($K_{Q_{i+1}}$, $\mathcal{RT}_{Q_{i+1}}$ and $K_{Q_{i-1}}$ are all verified) **then**
- 9: $M_i \leftarrow [M_{i-1}||K_{Q_{i-1}}]_{k_{Q_i}}$

Final Step:

- 10: p sends $M_{\ell-1}$ to $D \subseteq Q_\ell$ along with m .

RCP-II: RECEIVING PEER q

Initial Step:

- 1: **if** ($q \in Q_1$ receives $[p_{\text{ID}}|p_{\text{addr}}|\mathbf{key}|ts]$ from $p \in Q_1$) **then**
- 2: q checks that p 's request is legitimate and, if so, sends its signature share.

Intermediate Steps:

- 3: **if** ($q \in Q_i$ receives M_{i-1} from p) **then**
- 4: **for** $j = i - 1$ downto 1 **do**
- 5: q uses K_{Q_j} to verify $K_{Q_{j-1}}$.
- 6: Peer q uses K_{Q_1} to verify M_1 .
- 7: **if** verification is successful **then**
- 8: q sends $[K_{Q_{i-1}}]_{k_{Q_i}}$, $[\mathcal{RT}_{Q_{i+1}}]_{k_{Q_i}}$ and $[K_{Q_{i+1}}]_{k_{Q_i}}$ to p .

Figure 5.4: Pseudocode for RCP-II

Proof. First we prove the correctness of our protocol and, as before, we show that if p is correct and has not violated the rule set, at each step i of the protocol p either (1) establishes a valid M_i and receives the routing information for the next hop or (2) terminates the protocol by delivering m to all members of D . Our proof is by induction on i .

Base Case: Consider the initial step $i = 1$ where p communicates with the peers in its quorum $Q_p = Q_1$ about sending the message m . If p is correct and has not violated the rule set, upon receiving $[p_{\text{id}}|p_{\text{addr}}|\text{key}|ts]$ all correct peers will send their shares to p . Therefore, p is guaranteed to obtain M_1 by the majority invariant. Peer p already has the routing information for Q_2 ; therefore, the base case holds.

Inductive Hypothesis: Assume that at step $i < \ell$, p has obtained a correct M_i and routing information for Q_{i+1} .

Inductive Step: First assume that $i = \ell - 1$. Then, by the induction hypothesis, peer p possesses $M_{\ell-1}$ and the necessary routing information to send this signature and message m to $D \subseteq Q_{i+1}$; thus the protocol terminates correctly. Otherwise, assume $i < \ell - 1$; we consider step $i + 1$. Peer p sends M_i to a peer $q \in Q_{i+1}$ selected uniformly at random without replacement. By the inductive hypothesis, the contents of M_i are valid and p possesses the necessary routing information. If q is a Byzantine peer, then p 's request can fail and p can detect an invalid response using $K_{Q_{i+1}}$ obtained from the previous step. It is also possible that q is a correct but slow node and does not respond in a predefined time period. In this case, p re-issues its request to another randomly selected peer in Q_{i+1} ; eventually, one of selected correct peers will respond with $[K_{Q_i}]_{k_{Q_{i+1}}}$, $[\mathcal{RT}_{Q_{i+2}}]_{k_{Q_{i+1}}}$ and $[K_{Q_{i+2}}]_{k_{Q_{i+1}}}$ to p . Peer p will verify this information and create a valid M_{i+1} . Therefore, at this point p possesses a correct M_{i+1} and routing information for Q_{i+2} ; therefore, the induction holds.

Since RCP-II is a randomized algorithm, our costs are given in expectation. We assume the following: let X_i be a random variable denoting the time required for the i^{th} correct peer (note we condition on correctness) selected u.a.r without replacement by p to respond to p 's request. We assume that $Pr[X_i \leq \Delta] = c$ where $c > 0$ is some constant probability.

We now calculate loose upper bounds of the expected resource costs. In the first step, in communicating with Q_1 , peer p handles at most $2 \cdot s$ messages

and the round-trip latency is 1. Then for each step $i = 2, \dots, \ell - 1$, let Y_i be the random variable with value 1 if the i^{th} peer is both correct and responds within time Δ ; 0 otherwise. Then $Pr[Y_i = 1] \leq (1 - \epsilon) \cdot c$; for simplicity, set $\rho = (1 - \epsilon) \cdot c$ to be this probability of success. Let $Y = \sum_{i=1}^s Y_i$. The expected number of selections $E[Y]$ before p receives a response from a correct peer is at most:

$$\sum_{k=0}^s (1 - \rho)^k \cdot \rho \cdot (k + 1) = \rho \left(\sum_{k=0}^s (1 - \rho)^k \cdot k + \sum_{k=0}^s (1 - \rho)^k \right)$$

Therefore $E[Y] \leq \frac{1}{(1 - \epsilon) \cdot c}$ and including the last step, the expected latency is at most $\frac{\ell - 2}{(1 - \epsilon) \cdot c} + 2$. The ℓ^{th} step requires D messages and one hop. In terms of expected message complexity, since each step requires at most 2 messages and the last step requires $|D|$ messages, we can give a crude upper bound of $2s + \frac{2}{(1 - \epsilon) \cdot c} \cdot (\ell - 2) + |D|$. However, note that once p hears back from a node, any message from any other previously selected nodes in the current step can be easily ignored/filtered. Therefore, per step, p handles $\frac{1}{(1 - \epsilon) \cdot c} + 1$ messages. We can now give a more accurate upper bound of $2s + \frac{\ell - 2}{(1 - \epsilon) \cdot c} + (\ell - 2) + |D|$. Finally, while latency is measured in the number of communication rounds, the expected duration of time required for each intermediate round is $\frac{\Delta}{(1 - \epsilon) \cdot c}$.

In terms of the expected message complexity of a forwarding peer $q \notin D$ in a quorum along the lookup path, a correct peer chosen by p receives one message and sends one message. The probability that q is chosen is at most $1/((1 - \epsilon) \cdot s)$; therefore the expected message complexity for q is at most $2/((1 - \epsilon) \cdot s)$. \square

While latency is measured in communication rounds, the time for executing RCP-II depends on Δ and we discuss this briefly. Accounting for the response time incurred in the intermediate steps, p waits for at most time $\frac{\Delta}{(1 - \epsilon) \cdot c}$ per step in expectation as shown in Lemma 25. Since peer p will have knowledge of the response time distribution, p may optimize performance by selecting Δ so that $\frac{\Delta}{c}$ is minimized. Note that, for simplicity, our pseudocode for RCP-II does not address the case where none of the peers respond to p within Δ time. In this case, p should simply continue waiting until it receives a response.

Spamming Attacks: Due to the iterative nature of RCP-II, p sends more messages than other participating peers, but not to the degree seen in RCP-

I. Rather than make it expensive for p to perform robust communication, RCP-II uses two properties to deter spamming: (1) it is inexpensive for a peer to detect spam and (2) the congestion suffered by a peer is low since the number of messages is not magnified by the use of quorums.

To address our first point, p may launch as many robust communication operations as the rule set allows; p may even try to circumvent the rule set by directly sending to a correct peer q ; however, it is inexpensive for q to verify that the proof being sent is invalid. The operation terminates at that point since q will not reply. In contrast to the passive scenario of Section 5.4.4, q need not keep a history to judge the legitimacy of a request; it simply verifies the accompanying certificate.

Our second point, and a key difference between RCP-I and RCP-II, is that with RCP-II an operation incurs only expected $O(\ell)$ messages which compares favourably to a system *without a quorum topology*. Therefore, the congestion caused by such requests is not significantly magnified by the use of quorums which was a key concern regarding spamming.

Adversarial peers may misbehave in other ways with many of the same consequences and remedies as discussed in RCP-I. Even with a generous upper bound on the expiration of ts , the congestion p can cause with a replay attack is again limited since only p can use the certificate. A notable attack, unique to RCP-II, occurs when a faulty peer gives p stale routing table information. Since entries are signed and time stamped, we are guaranteed that in the fairly recent past, the location indicated by the stale information was indeed correct. This fact, coupled with the standard assumption that ID collisions do not occur, guarantees that the adversary *cannot* engineer a situation where requests are forwarded to a faulty peer. Consequently, the impact of this attack is limited. The search path may be slightly lengthened by forwarding to an older location. Alternatively, stale information may point to a peer that no longer exists or is not the correct recipient, which forces p to backtrack one hop. These cases are handled easily, but for ease of exposition, they are not treated in our pseudocode in Figure 5.4. Routing integrity is not compromised and, since routing tables can be signed periodically every several minutes without significant CPU cost (see Section 5.7), the impact of such an attack is negligible.

5.6.2 The Join Protocols and Membership Updates

For the sake of being self-contained, we describe how a peer would join our system. This first involves a discussion of a result by Awerbuch and Scheideler [13] which allows a DHT to be robust even if the number of join and leave events is polynomial in the size of the network n . More precisely, within a window of time, the adversary may opt to insert a Byzantine peer (assuming the total number of Byzantine peers in the system does not exceed the allotted amount) or remove a Byzantine peer from the system. An adversary may attempt to gain a majority of Byzantine peers in a targeted quorum Q by having one of its peers q' join the system. If q' 's location in the DHT does not allow it to become part of Q , then the adversary removes q' and has it rejoin for another attempt. By repeating this process with several peers, the adversary can eventually obtain a majority in Q , at which point the security of the system is compromised. The protocol for defending against such attacks is the *cuckoo rule* developed by Awerbuch and Scheideler [13]. We assume the identifier space of the DHT is normalized to be $[0,1)$. For any interval $I \subset [0, 1)$, the cuckoo rule maintains two invariants. The first is the *balancing invariant* which guarantees that I contains $\Theta(|I| \cdot n)$ peers. The second is the *majority invariant* which guarantees the majority of peers in I are correct. The authors show that for $|I| = \Theta(\log(n)/n)$ both invariants can be maintained with high probability over n^c join and leave operations, where c is a constant that can be tuned according to the parameters of the protocol. It follows that the peers in I can form a quorum.

It is important to understand the resource costs of the cuckoo rule which functions as follows. The ring $[0, 1)$ is assumed to be broken into disjoint segments of constant length k/n for some constant k . Each segment is called a *k-region* and $R_k(x)$ denotes the unique k -region containing x . When a peer p joins the network, it is assigned a random identifier $x \in [0, 1)$ and placed in this location on the ring. All nodes in $R_k(x)$ are evicted from their locations and placed into new locations chosen uniformly and independently at random from $[0, 1)$. Figure 5.5 illustrates these operations.

The node placements required by the cuckoo rule can be executed by having quorums use robust communication in order to inform each other about the arrival of the evicted nodes at their new locations. Once a quorum Q_i knows about the presence of a recently evicted node q , all correct members of Q_i update their membership lists, share IP addresses, and aid q in setting up

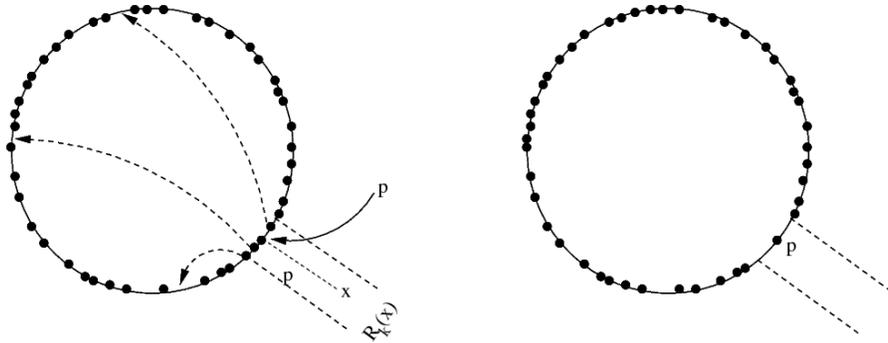


Figure 5.5: An illustration of the cuckoo rule. (Left) Peer p is placed in its random location x . All peers in the k -region, $R_k(x)$ denoted by dashed lines, are assigned random locations in $[0, 1)$. (Right) After the cuckoo rule is executed, peer p is the only peer in $R_k(x)$.

any required links (i.e. such as finger links in Chord). A detailed discussion of how this can be done is presented in [53]. Furthermore, the issue of efficiently generating random numbers for use with the cuckoo rule is addressed by Awerbuch and Scheideler [12]. We also note that resilience to a more challenging attack, where the adversary is able to force correct peers to leave the system temporarily, is addressed by an extension of the cuckoo rule given by Awerbuch and Scheideler [14]; it also seems possible to implement this rule in concert with our communication protocols. We finish our discussion of a join protocol by discussing the steps necessary for maintaining DKG and the consequent cost of membership changes:

RCP-I: Consider a quorum Q_i to which a new peer is added. The membership update protocol of DKG [85] is executed to redistribute the shares of the public/private quorum key pair over all members of Q_i . In the process, the individual public/private key shares are also updated. Notably, *no other quorums are affected by this process* as the quorum key pair remains the same and the individual key shares need only be known to members of Q_i . When a peer leaves Q_i , the departure can be treated as a crash and so long as the number of crashes does not exceed the crash limit f , the DKG (share renewal) protocol need not be executed. We use this to associate the system churn rate to DKG session time. Note that the adversary may crash some of its t nodes, and in principle, the system can handle $t + f$ node leaves. How-

ever, we cannot associate these additional t crashes with the system churn due to the inherent arbitrary nature of Byzantine peers.

RCP-II: When a peer q joins Q_i , the DKG protocol needs to be executed as in the case of RCP-I; however, there are additional costs due to the need to update and re-sign the routing table information. In particular, not only do the peers in Q_i need to update and have signed their routing table information to reflect the addition of q , all quorums to which Q_i is linked under the quorum topology also need to update and re-sign their routing table information; note that this *does not require any revocation* since the public key does not change. Therefore, a join event under this scheme *does affect other quorums*. When a peer leaves Q_i , DKG may be required as in the case of RCP-I. However, routing table information for Q_i and the quorums to which it links must again update and re-sign their routing table information. Therefore, while RCP-II reduces message complexity, the cost of join/leave operations is higher in comparison to RCP-I.

5.7 Microbenchmarks and Performance

We examine the performance of DKG and our two protocols through microbenchmarks executed on the PlanetLab platform [122]. Based on our experimental results and known churn rates, we propose parameters for DHTs using our protocols.

5.7.1 Implementation and Microbenchmarks

The DKG protocol is a crucial component of our protocols since it is required to initiate a threshold signature system in a quorum and to securely manage membership changes. Here, a C++ implementation [86] is used to measure the performance of DKG. The experimental measurements reported in Table 5.1 and those regarding the threshold BLS signatures were executed on PlanetLab. While this author had input into certain design aspects of the DKG experiment that yielded the values in Table 5.1, the implementation of the BLS signatures and the execution of the DKG experiments were performed by Dr. Aniket Kate; these results are presented in more detail in [169]. In this thesis, we use the results of these experiments to derive back-of-the-envelope calculations with respect to the performance of our protocols.

Distributed Key Generation: We test the DKG implementation for quorum sizes $s = 10, 15, 20, 25, 30$ and present median completion times and median CPU usage in Table 5.1 along with 95% one-sided confidence intervals. We describe our experimental setup in the context of [64, 93]. Our experiments are terminating and conducted via the method of independent replications. A single replication consists of s individual observations each corresponding to the time required for a participating peer in the quorum to complete the DKG protocol; there are 10 replications for each s value. For each s value, the PlanetLab machines used are chosen from around the world with roughly 64% located in North America, 20% located in Asia and the remaining 16% located in Europe. Using independent replications, an unbiased sample point estimator for variance is calculated and used to obtain our one-sided confidence intervals using the t -distribution.

The median completion periods vary from roughly 10 seconds for $s = 10$ to roughly 69 seconds for $s = 30$. Notably, the bulk of this latency is due to network delay whereas the required CPU time is far smaller than the completion periods. In the next subsection, we examine the feasibility of these completion periods. Our DKG experiments are set up so that correctness is guaranteed so long as at most 30% of the peers may crash and 10% of the peers may be Byzantine. While we can tolerate any fraction of Byzantine peers less than $1/3$, we use these numbers since in many practical scenarios we expect the fraction of Byzantine faults to be less than 10% and modest compared to the fraction of crash failures. In each of our replications, the pseudorandom values are generated using the well-known Number Theory Library (NTL) and Pairing-Based Cryptography (PBC) library.

RCP-I and RCP-II: For our RCP-I and RCP-II experiments, we set $s = 30$, $t = 3$, and $f = 10$. We conduct terminating experiments again via the method of independent replications where each of the 5 replications consists of 30 observations. In RCP-I, a node requires an average of 0.14 ± 0.0075 seconds (95% one-sided confidence interval) to obtain a threshold signature from a quorum, if all of the obtained signature shares are correct. The average execution time increases to 0.23 ± 0.015 seconds (95% one-sided confidence interval) in the case of a share corruption attack. Extrapolating to a path length ℓ , an operation should take between $0.14 \cdot \ell \pm 0.0075 \cdot \ell$ or $0.23 \cdot \ell \pm 0.015 \cdot \ell$ seconds on average. For a DHT with 10^5 nodes, the average total time for RCP-I is then 2.8 ± 0.15 to 4.6 ± 0.3 seconds with $\ell = 20$.

Table 5.1: Median values of DKG completion time and CPU time per node for various s values.

s	t	f	Time (seconds)	CPU Seconds/Node
10	1	3	8.59 ± 2.50	1.31 ± 0.13
15	2	4	15.88 ± 12.72	2.11 ± 0.14
20	2	6	26.38 ± 0.89	2.37 ± 0.19
25	3	7	46.55 ± 5.27	6.12 ± 0.54
30	3	10	61.82 ± 4.31	5.46 ± 0.49

In RCP-II, a node takes 0.042 ± 0.014 seconds (95% one-sided confidence interval) on average to obtain the required signed public keys and the signed routing information from a correct peer. A single signature verification takes negligible time; however, for completeness we report the average value of 0.0045 ± 0.0028 seconds (95% one-sided confidence interval). The median latency value over *all* pairs of PlanetLab nodes is roughly 0.08 seconds [42]; that is, $\Delta = 0.08$ seconds for $c = 0.5$. With a chain of signed public keys of length ℓ , the total communication time is $0.14 \pm 0.0075 + (0.042 \pm 0.014) \cdot (\ell - 1) + \frac{\Delta \cdot (\ell - 2)}{c \cdot (1 - \epsilon)} + (0.0045 \pm 0.0028) \cdot \frac{\ell(\ell - 1)}{2}$ which for 10% Byzantine peers, is 4.94 ± 1.60 seconds in expectation for $\ell = 20$. To a first approximation, the execution times of our protocols seem quite reasonable.

System Load: We address the issue of system load under the assumption that signature verification is the most significant computational operation. We make back-of-the-envelope calculations to obtain the expected order of magnitude for our performance figures. For RCP-I, from the above discussion, each signature verification takes 0.0045 ± 0.0028 seconds; thus, the total CPU time required per execution is $(0.0045 \pm 0.0028) \cdot \ell \cdot (1 + s + s^2)$; this includes the costs due to share corruption attacks. For $\ell = 20$ and $s = 30$, this value is 74.48 ± 52.14 CPU seconds, spread out over 600 nodes. Therefore, the number of executions of RCP-I that can be started per second on average when $n = 10^5$ is roughly 10^3 ; note this rate value is for *the entire system*. Now, if no share corruption attacks occur, the total CPU time required per execution becomes $(0.0045 \pm 0.0028) \cdot \ell \cdot (1 + s)$ which, for the same parameter values, is 2.5 ± 1.76 CPU seconds. This implies that $4 \cdot 10^4$ executions can be started per second on average in the entire system. For RCP-II, the total CPU time required for execution is given by $(0.0045 \pm 0.0028) \cdot \left(\ell + \frac{(\ell - 1) \cdot \ell}{2 \cdot (1 - \epsilon)} \right)$

Table 5.2: The expected number of seconds before a quorum experiences a membership change (r_Q).

s	10			15			20			25			30		
n_Q	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
r_Q	526	351	175	350	234	117	263	132	88	210	140	70	175	87	58

which, for the same parameters and $\epsilon = 1/10$ is 1.040 ± 0.65 CPU seconds on average. Therefore, approximately 10^5 executions can be started per second on average in the entire system.

5.7.2 Analysis and Discussion

As mentioned in Section 5.4.1, important questions remain with regards to translating theoretical results to a practical setting. In particular, two quantities of interest are the size of quorums, s , and the number of quorums to which each peer belongs, n_Q . Unfortunately, pinning down these quantities is non-trivial. Only asymptotic analysis is present in the literature. Furthermore, it is not a simple case of substituting hard numbers because s depends on a number of parameters: (1) the exact guarantees being made, (2) algorithms for quorum maintenance, (3) the tools of analysis (i.e. form of Chernoff bounds used) and many more. Evaluating these parameters is outside the scope of this work. Instead, we assume a range of values for s and n_Q . As our protocols appear to be the most efficient to date, the following results illuminate what currently seems possible in practice.

System Churn and DKG: We now return to the issue of system churn which was discussed earlier in Section 5.5. A common metric for measuring the degree of churn is *session time*: the time between when a node joins the network and when it departs [129]. As discussed in Section 5.4.4, we make the standard assumption that the cost of joining the network is large enough so as to prevent the adversary from substantially increasing the rate of churn through rapid rejoin operations.

Part I - An Argument for Batching: Investigations have yielded differing measurements for median session times. The Kazaa system was found to have a median session time of 144 seconds [68]. In the Gnutella and Napster networks, the median session time was measured to be approximately 60 minutes [136]. In the KAD DHT, 155 minutes was the measured median [146]. A study of the Skype P2P network yielded a median session time of 5.5 hours for super-peers [65]. Here, we temporarily assume a median session time of 60 minutes and a standard Poisson model of peer arrivals/departures as in [98,129]. To calculate churn rate, r (number of arrivals/departures per second), based on the median session time t_{med} (in seconds), we use the formula of [129]: $r = (n \cdot \ln 2)/t_{med}$. For $n = 10^5$ and $t_{med} = 3600$ seconds, $r \approx 19$. Assuming that join and departure events occur independently of each other, Table 5.2 gives the expected number of seconds, r_Q , at which point a quorum will undergo a membership change when each peer belongs to n_Q quorums. Our choice of $n_Q \leq 3$ is based upon the reasonable assumption that overlap occurs only with neighboring quorums in the ID space.

In several cases, the r_Q values are less than or fairly close to the corresponding median DKG completion times in Table 5.1. Therefore, a quorum may not be able to execute DKG often enough to accommodate each membership change. However, join operations can be queued and performed in batches. Executing DKG for a batch of joins does not increase the message complexity and message size increases only linearly in the batch size (see [85]). Therefore, batching can mitigate the effects of churn and it seems plausible that peers would tolerate some delay in joining in exchange for security.

Part II - Batching and the Security Threshold: Batching join events improves performance; however, many peers might depart a quorum before a new batch is added, thus violating the security threshold. Hence, we are interested in the median session time value required such that this is not *likely* to occur. Based on Table 5.1 for $s = 20$ and $n_Q = 1$, DKG completes within roughly 26 seconds. The number of departures a quorum can suffer while not exceeding the crash limit is $f = 6$. If Byzantine peers depart, more crashes are tolerable; however, identifying such events is impossible, so we assume the worst case of $f = 6$. Assuming DKG executes every $r_{DKG} = 1200$ seconds, we seek the median session time such that at most 6 peers depart the system within 1226 seconds. With $n/s = 5000$ quorums in the system, each experiencing 6 departures within 1226 seconds, the system churn rate is

Table 5.3: Median session times (in hours) derived from values for s , n_Q and r_{DKG} (in hours).

s	10			15			20		
r_{DKG}	0.17			0.25			0.33		
n_Q	1	2	3	1	2	3	1	2	3
t_{med}	0.39	0.78	1.17	0.67	1.33	2.00	0.79	1.57	2.36
	25			30					
	0.42			0.50					
	1	2	3	1	2	3			
	1.07	2.15	3.22	1.08	2.16	3.24			

roughly $r = 25$. This gives $t_{med} = 2832$ or, equivalently, 47 minutes. Therefore, with this t_{med} , we expect the system to remain secure, and a quorum only spends $26/1226 = 2.1\%$ of the time executing DKG. We note that this is not a completely rigorous argument as we are considering median session times. However, since churn is Poisson distributed, the probability of exceeding 6 departures within 1226 seconds quickly decreases. In order to obtain more rigorous guarantees on security, it seems likely that larger quorum sizes are necessary; regardless, our calculations provide a first-approximation of the fraction of time a quorum spends executing the DKG protocol.

We can decrease the required median session times by decreasing r_{DKG} ; however, the percentage of time spent on DKG increases. Such tuning would depend on the desired system performance, the application, s , and n_Q . Table 5.3 gives session time calculations for other values of s , r_{DKG} and n_Q . Required session times increase with s . Notably, for $s = 30$ and $n_Q = 1$, t_{med} does not far exceed the 60 minutes in [136]. As n_Q increases, the required session times grow linearly. However, our maximum of 3.24 hours is still less than t_{med} measured for super-peers in the Skype network [65]. We tentatively conclude that our protocols can be deployed in applications where session times range from 10 minutes to a few hours and that such applications currently exist.

Chapter 6

Final Remarks

In this thesis, we have presented attack-resistant algorithms for achieving resource-efficient communication in both wired and wireless networks. Our algorithms achieve improvements over the previous best known results. We conclude this thesis by briefly summarizing our main contributions and posing some open problems as possible avenues for future work.

Our first major result addressed a simple single-hop communication scenario involving a single sender, a single receiver, and an adversary that can interfere with the shared communication channel. Our results demonstrate that an adversary is required to expend an asymptotically greater amount of resources than that spent by a correct sender and receiver. We then extended this work to demonstrate the utility of our results in mitigating attacks in WSNs when a jamming adversary is present, and in the wired client-server model when an application level DoS attack occurs. In terms of future work, there are several interesting questions that remain open. Is our result on favourable communication the best possible? It would be valuable to obtain a lower bound on the 3-Player Scenario. It also seems that our approach would prove useful for gossiping and epidemic protocols where the propagation of a message in the early stages is critical to its dissemination throughout the network; the investigation of such applications is a topic for future work.

With regards to WSNs, we designed new algorithms for addressing the problem of reducing energy consumption when Byzantine faults are present. To this end, we considered a second single-hop communication scenario involving multiple (possibly faulty) senders and a single receiver. Our abstrac-

tion of this problem is a novel data streaming problem which we call the Bad Santa problem. We have shown how our results on this problem can be applied to the problem of reliable broadcast in WSNs in a grid network. Our algorithms for reliable broadcast on a grid consume significantly less power than any other algorithms for this problem of which we are aware. Several open problems remain including: Can we close the gap between the upper and lower-bound for the multi-round Bad Santa problem? Can we achieve more energy efficiency for the optimal number of Byzantine faults? Can we tolerate more faults for the fail-stop model and still be energy efficient? Can we tolerate more faults in the unknown source and message time scenario? Can we generalize our techniques to other topologies? Are there other applications for the Bad Santa problem both in and outside the domain of WSNs?

In the domain of peer-to-peer networks, we have provided two new robust communication protocols that leverage cryptographic techniques to improve asymptotically on the message complexity of previous results. Our experimental work suggests that our protocols are practical for a number of application scenarios. In terms of future work, the performance of a complete system is an important open question — the quorum topology chosen is crucial and optimizing this in practice requires further study. While we assume that peer arrivals and departures obey a Poisson distribution, the actual distribution may be different. This has implications for our arguments involving execution of the DKG protocol and batching, and more experimental work would be valuable. The cuckoo rule deserves further attention since it likely requires substantial overhead. Optimizing its performance and reconciling it with our batching mechanism for peer joins is important. Finally, while we focus on DHTs, our results may apply to other P2P designs and more general settings where groups of machines, some with untrustworthy members, must communicate; it would be of interest to identify such applications.

Bibliography

- [1] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. Fault-Scalable Byzantine Fault-Tolerant Services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, pages 59–74, 2005. 109
- [2] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, available, and reliable storage for incompletely trusted environment. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–14, 2002. 109
- [3] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38:393–422, 2002. 1, 3
- [4] Dan Alistarh, Seth Gilbert, Rachid Guerraoui, Zarko Milosevic, and Calvin Newport. Securing Your Every Bit: Reliable Broadcast in Byzantine Wireless Networks. In *Proceedings of the 22nd Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 50–59, 2010. 28, 39, 44, 45, 50
- [5] Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–29, 1996. 71
- [6] Hind Alwan and Anjali Agarwal. A Survey on Fault Tolerant Routing Techniques in Wireless Sensor Networks. In *Proceedings of the 3rd In-*

ternational Conference on Sensor Technologies and Applications, pages 366–371, 2009. 3

- [7] Giuseppe Anastasi, A. Falchi, Andrea Passarella, Marco Conti, and Enrico Gregori. Performance Measurements of Motes Sensor Networks. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 174–181, 2004. 56
- [8] Nils Aschenbruck, Elmar Gerhards-Padilla, and Peter Martini. Simulative Evaluation of Adaptive Jamming Detection in Wireless Multi-hop Networks. In *Proceedings of the 30th International Conference on Distributed Computing Systems Workshops*, pages 213–220, 2010. 28
- [9] James Aspnes, Navin Rustagi, and Jared Saia. Worm versus alert: Who Wins in a Battle for Control of a Large-Scale Network? In *Proceedings of the 11th International Conference On Principles Of Distributed Systems (OPODIS)*, pages 443–456, 2007. 106
- [10] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant Authentication with Client Puzzles. In *8th International Workshop on Security Protocols*, pages 170–177, 2000. 29
- [11] Baruch Awerbuch, Andrea Richa, and Christian Scheideler. A Jamming-Resistant MAC Protocol for Single-Hop Wireless Networks. In *Proceedings of the 27th Symposium on the Principles of Distributed Computing (PODC)*, pages 45–54, 2008. 28, 39, 45
- [12] Baruch Awerbuch and Christian Scheideler. Robust Random Number Generation for Peer-to-Peer Systems. In *Proceedings of the 10th International Conference On Principles of Distributed Systems (OPODIS)*, pages 275–289, 2006. 3, 19, 20, 106, 110, 113, 133
- [13] Baruch Awerbuch and Christian Scheideler. Towards a Scalable and Robust DHT. In *Proceedings of the 18th Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 318–327, 2006. 3, 19, 20, 106, 107, 110, 111, 112, 113, 132
- [14] Baruch Awerbuch and Christian Scheideler. Towards Scalable and Robust Overlay Networks. In *Proceedings of the 6th International work-*

- shop on Peer-To-Peer Systems (IPTPS)*, 2007. 3, 19, 20, 106, 110, 113, 116, 133
- [15] Joe Bardwell. Converting Signal Strength Percentage to dBm Values, 2002. 39
- [16] Salman A. Baset and Henning Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proceedings of the 30th International Conference on Computer Communications (INFOCOM)*, pages 1–11, 2006. 2
- [17] Emrah Bayraktaroglu, Christopher King, Xin Liu, Guevara Noubir, Rajmohan Rajaraman, and Bishal Thap. On the Performance of IEEE 802.11 under Jamming, 2008. 28
- [18] BBC. Anonymous Hacktivists Say Wikileaks War to Continue. www.bbc.co.uk/news/technology-11935539, 2010. 22
- [19] Marin Bertier, Anne-Marie Kermarrec, and Guang Tan. Brief Announcement: Reliable Broadcast Tolerating Byzantine Faults in a Message-Bounded Radio Network. In *Proceedings of the 22nd International Symposium on Distributed Computing (DISC)*, pages 516–517, 2008. 13, 28
- [20] Marin Bertier, Anne-Marie Kermarrec, and Guang Tan. Message-Efficient Byzantine Fault-Tolerant Broadcast in a Multi-Hop Wireless Sensor Network. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 408–417, 2010. 28, 44, 45, 50
- [21] Vartika Bhandari and Nitin H. Vaidya. On Reliable Broadcast in a Radio Network. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 138–147, 2005. 13, 27, 28, 44, 63, 64, 66, 70, 86, 100, 103
- [22] Vartika Bhandari and Nitin H. Vaidya. On Reliable Broadcast in a Radio Network: A Simplified Characterization. Technical report, CSL, UIUC, May 2005. xiii, 13, 14, 15, 16, 27, 28, 44, 45, 46, 47, 48, 53, 63, 64, 65, 66, 70, 71, 79, 81, 82, 84, 85, 86, 90, 93, 98, 100, 101, 102, 103

- [23] Vartika Bhandari and Nitin H. Vaidya. Reliable Broadcast in Wireless Networks with Probabilistic Failures. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 715–723, 2007. 13, 28, 44, 70, 102
- [24] Vartika Bhandhari, Jonathan Katz, Chiu-Yuen Koo, and Nitin Vaidya. Reliable Broadcast in Radio Networks: The Bounded Collision Case. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 258 – 264, 2006. 28, 39, 44, 45, 50, 63, 70, 100
- [25] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Proceedings of the International Conference on Practice and Theory in Public Key Cryptography (PKC)*, pages 31–46, 2003. 115
- [26] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 416–432, 2003. 127
- [27] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. In *Proceedings of the Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 514–532, 2001. 115
- [28] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine Resilient Random Membership Sampling. In *Proceedings of the 27th Symposium on the Principles of Distributed Computing (PODC)*, pages 145–154, 2008. 110
- [29] David Boyle and Thomas Newe. Security Protocols for Use with Wireless Sensor Networks: A Survey of Security Architectures. In *Proceedings of the 3rd International Conference on Wireless and Mobile Communications*, page 54, 2007. 3
- [30] Christian Cachin and Jonathan A. Poritz. Secure Intrusion-tolerant Replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 167–176, 2002. 109

- [31] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, pages 299–314, 2002. 3, 110, 118
- [32] Miguel Castro and Barbara Liskov. Byzantine Fault Tolerance Can Be Fast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 513–518, 2001. 109
- [33] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002. 112
- [34] Xiangqian Chen, Kia Makki, Kang Yen, and Niki Pissinou. Sensor network Security: A Survey. *IEEE Communications Surveys & Tutorials*, 11(2):52–73, 2009. 3
- [35] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport, and Tina Nolte. Consensus and Collision detectors in Wireless Ad Hoc Networks. In *Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 197 – 206, 2005. 71
- [36] Nathan Coopriider, Will Archer, Eric Eide, David Gay, and John Regehr. Efficient Memory Safety for TinyOS. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems*, pages 205–218, 2007. 103
- [37] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Space- and Time-Efficient Deterministic Algorithms for Biased Quantiles Over Data Streams. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 263–272, 2006. 71
- [38] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 177–190, 1999. 109

- [39] John Cox. For Wireless Sensor Nets, Reality Sets In. <http://features.techworld.com/networking/1863/for-wireless-sensor-nets-reality-sets-in/>, 2005. 22
- [40] Alexandra Czarlinska and Deepa Kundur. Wireless Image Sensor Networks: Event Acquisition in Attack-Prone and Uncertain Environments. *Multidimensional Systems and Signal Processing*, 20(2):135–164, 2009. 105
- [41] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–215, 2001. 18
- [42] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. Designing a DHT for Low Latency and High Throughput. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pages 85–98, 2004. 136
- [43] Eric Demaine, Alejandro López-Ortiz, and Ian Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 348–360, 2002. 71
- [44] Jing Deng, Pramod K. Varshney, and Zygmunt J. Haas. A New Backoff Algorithm for the IEEE 802.11 Distributed Coordination Function. In *Communication Networks and Distributed Systems Modeling and Simulation*, pages 215–225, 2004. 39
- [45] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, Fabian Kuhn, and Calvin Newport. The Wireless Synchronization Problem. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing, PODC '09*, pages 190–199, 2009. 28
- [46] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Gossiping in a Multi-channel Radio Network: An Oblivious Approach to Coping with Malicious Interference. In *Proceedings of the 21st International Symposium on Distributed Computing (DISC)*, pages 208–222, 2007. 28

- [47] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Secure Communication over Radio Channels. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 105–114, 2008. 28
- [48] John R. Douceur and Jon Howell. Byzantine Fault Isolation in the Farsite Distributed File System. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006. 109
- [49] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a Million User DHT. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 129–134, 2007. 1, 106
- [50] Laura Marie Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of the 20th IEEE International Conference on Computer and Communications (INFOCOM)*, pages 1548–1557, 2001. 61
- [51] Paul Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1987. 114
- [52] Amos Fiat and Jared Saia. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 94–103, 2002. 3, 19
- [53] Amos Fiat, Jared Saia, and Maxwell Young. Making Chord Robust to Byzantine Attacks. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 803–814, 2005. 3, 19, 106, 107, 110, 111, 112, 113, 116, 118, 133
- [54] Jason Franklin, Vern Paxson, Adrian Perrig, and Stefan Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *14th ACM Conference on Computer and Communications Security*, pages 375–388, 2007. 5, 57, 60
- [55] Saurabh Ganeriwal, Christina Pöpper, Srdjan Čapkun, and Mani B. Srivastava. Secure Time Synchronization in Sensor Networks. *ACM Transactions on Information and System Security*, 11(23), 2008. 38

- [56] Lee Garber. Denial-of-Service Attacks Rip the Internet. *Computer*, 33(4):12–17, 2000. 29
- [57] Leszek Gasieniec, Erez Kantor, Dariusz R. Kowalski, David Peleg, and Chang Su. Time Efficient k -Shot Broadcasting in Known Topology Radio Networks. *Distributed Computing*, 21(2):117–127, 2008. 71
- [58] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing Data Privacy with Self-Destructing Data. In *Proceedings of the USENIX Security Symposium*, pages 299–315, 2009. 2, 106
- [59] Johannes Gehrke and Samuel Madden. Query Processing in Sensor Networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004. 105
- [60] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust Threshold DSS Signatures. In *Advances in Cryptology - EUROCRYPT, Lecture Notes in Computer Science*, pages 354–371, 1996. 115
- [61] Seth Gilbert, Rachid Guerraoui, Dariusz Kowalski, and Calvin Newport. Interference-Resilient Information Exchange. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 2249–2257, 2009. 28
- [62] Seth Gilbert, Rachid Guerraoui, and Calvin C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. In *Proceedings of the International Conference On Principles Of Distributed Systems (OPODIS)*, pages 215–229, 2006. 3, 28, 39, 71
- [63] Virgil D. Gligor. Guaranteeing Access in Spite of Distributed Service-Flooding Attacks. In *Security Protocols Workshop*, 2003. 29
- [64] David Goldsman and Gamze Tokol. Output Analysis: Output Analysis Procedures for Computer Simulations. In *Proceedings of the 32nd Conference on Winter Simulation*, pages 39–45, 2000. 135
- [65] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006. 138, 139

- [66] Sudipto Guha and Andrew McGregor. Approximate Quantiles and the Order of the Stream. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 273–279, 2006. 71
- [67] Sudipto Guha and Andrew McGregor. Lower Bounds for Quantile Estimation in Random-Order and Multi-Pass Streaming. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming*, 2007. 71
- [68] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 314–329, 2003. 138
- [69] Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh Venkatesh. DoS Protection for Reliably Authenticated Broadcast. In *Proceedings of the 11th Networks and Distributed System Security Symposium (NDSS)*, 2004. 58
- [70] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. One Hop Lookups for Peer-to-Peer Overlays. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems (HotOS IX)*, pages 2–2, 2003. 110
- [71] Nicholar Harvey, Michael Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003. 3
- [72] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, pages 3005–3014, 2000. 38
- [73] Monika Henzinger, Prabhaker Raghavan, and Sridar Rajagopalan. Computing on Data Streams. Technical Report SRC-TN-1998-011, Digital Systems Research Center, 1998. 71

- [74] Ted Herman and Sébastien Tixeuil. A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks. *Algorithmic Aspects of Wireless Sensor Networks*, 3121:45–58, 2004. 105
- [75] Kristen Hildrum and John Kubiawicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of the 17th International Symposium on Distributed Computing*, pages 321–336, 2004. 19, 106, 107, 110
- [76] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, 2000. 61
- [77] Akira Ichimura and Maiko Shigeno. A New Parameter for a Broadcast Algorithm with Locally Bounded Byzantine Faults. *Information Processing Letters*, 110(12-13):514–517, 2010. 17
- [78] Hao Jiang and Constantinos Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM SIGCOMM Computer Communication Review*, 32:75–88, 2002. 127
- [79] Håvard Johansen, André Allavena, and Robbert van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 3–13, 2006. 106, 110
- [80] Ari Juels and John Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Networks and Distributed Security Systems (NDSS)*, pages 151–165, 1999. 29
- [81] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003. 3
- [82] Apu Kapadia and Nikos Triandopoulos. Halo: High-Assurance Locate for Distributed Hash Tables. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008. 110

- [83] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 162–175, 2004. 39
- [84] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, 2002. 3
- [85] Aniket Kate and Ian Goldberg. Distributed Key Generation for the Internet. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, pages 119–128, 2009. 115, 133, 138
- [86] Aniket Kate and Ian Goldberg. Asynchronous Distributed Private-Key Generators for Identity-Based Cryptography. In *Proceedings of the 7th Conference on Security and Cryptography for Networks (SCN)*, pages 436–453, 2010. 134
- [87] Valerie King, Cynthia Phillips, Jared Saia, and Maxwell Young. Sleeping on the Job: Energy-Efficient and Robust Broadcast for Radio Networks. In *Proceedings of the 27th ACM symposium on Principles of Distributed Computing (PODC)*, pages 243–252, 2008. 11, 13, 28, 37, 72, 74, 75
- [88] Valerie King, Cynthia Phillips, Jared Saia, and Maxwell Young. Sleeping on the Job: Energy-Efficient and Robust Broadcast for Radio Networks. *Accepted to Algorithmica*, 2010. 11, 13, 28, 37, 44, 46
- [89] Valerie King, Jared Saia, and Maxwell Young. Conflict on a Communication Channel. In *Proceedings of the 30th Symposium on Principles of Distributed Computing (PODC)*, pages 277–286, 2011. 11
- [90] Chiu-Yuen Koo. Broadcast in Radio Networks Tolerating Byzantine Adversarial Behavior. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 275–282, 2004. 13, 14, 16, 17, 27, 28, 44, 45, 50, 52, 63, 64, 66, 70, 77, 90, 100
- [91] Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Fault-Tolerant Broadcasting in Radio Networks. *Journal of Algorithms*, 39(1):47–67, 2001. 71

- [92] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Westley Weimer Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000. 109
- [93] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet Simulation Studies: The Incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9:50–61, 2005. 135
- [94] Yee Wei Law, Jeroen Doumen, and Pieter Hartel. Survey and Benchmark of Block Ciphers for Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 2(1):65–93, 2006. 39
- [95] HangRok Lee, YongJe Choi, and HoWon Kim. Implementation of TinyHash based on Hash Algorithm for Sensor Network. In *World Academy of Science, Engineering and Technology (WASET)*, pages 135–139, 2005. 103
- [96] Michael Lesk. The New Front Line: Estonia under Cyberassault. *IEEE Security and Privacy*, 5(6):76–79, 2007. 5, 57
- [97] Yuan Li, Wei Ye, and John Heidemann. Energy and Latency Control in Low Duty Cycle MAC Protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 676–682, 2005. 38
- [98] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *PODC*, pages 233–242, 2002. 138
- [99] Guolong Lin and Guevara Noubir. On Link Layer Denial of Service in Data Wireless LANs. *Wireless Communications & Mobile Computing*, 5(3):273–284, 2005. 28
- [100] Donggang Liu and Peng Ning. Multi-Level μ TESLA: Broadcast Authentication for Distributed Sensor Networks. *ACM Transactions in Embedded Computing Systems*, 3:800–836, 2004. 39

- [101] Xin Liu, Guevara Noubir, Ravi Sundaram, and San Tan. SPREAD: Foiling Smart Jammers using Multi-layer Agility. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 2536–2540, 2007. 28
- [102] Mario Livio. *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number*. Broadway, 2003. 9
- [103] Ewen MacAskill. WikiLeaks Website Pulled by Amazon after U.S. Political Pressure. www.guardian.co.uk/media/2010/dec/01/wikileaks-website-cables-servers-amazon, 2010. 22
- [104] Dominic Meier, Yvonne Anne Pignolet, Stefan Schmid, and Roger Wattenhofer. Speed Dating Despite Jammers. In *Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 1–14, 2009. 28, 43
- [105] Prateek Mittal and Nikita Borisov. ShadowWalker: Peer-to-peer Anonymous Communication using Redundant Structured Topologies. In *Proceedings of the 16th ACM conference on Computer and communications security (CCS)*, pages 161–172, 2009. 110
- [106] Michael Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California at Berkeley, 1996. 124, 125
- [107] William G. Morein, Angelos Stavrou, Debra L. Cook, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. Using Graphic Turing Tests to Counter Automated DDoS Attacks against Web Servers. In *Proceedings of the 10th ACM International Conference on Computer and Communications Security*, pages 8–19, 2003. 29
- [108] Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou. A Survey on Jamming Attacks and Countermeasures in WSNs. *IEEE Communications Surveys & Tutorials*, 11(4):42–56, 2009. 3, 9, 22
- [109] Ian Munro and Mike Paterson. Selection and Sorting with Limited Storage. *Theoretical Computer Science*, pages 315–323, 1980. 71

- [110] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc., 2005. 71
- [111] Arjun Nambiar and Matthew Wright. Salsa: A Structured Approach to Large-Scale Anonymity. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, pages 17–26, 2006. 110, 118
- [112] Moni Naor and Udi Wieder. A Simple Fault Tolerant Distributed Hash Table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 88–97, 2003. 3, 19, 20, 106, 107, 110, 111, 112, 113, 118
- [113] Vishnu Navda, Aniruddha Bohra, Samrat Ganguly, and Dan Rubenstein. Using Channel Hopping to Increase 802.11 Resilience to Jamming Attacks. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 2526–2530, 2007. 28
- [114] Dragoş Niculescu. Interference Map for 802.11 Networks. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 339–350, 2007. 39
- [115] Felix Oberholzer-Gee and Koleman S. Strumpf. The Effect of File Sharing on Record Sales: An Empirical Analysis. *Journal of Political Economy*, 115:1–42, 2007. 2
- [116] Vasileios Pappas, Dan Massey, Andreas Terzis, and Lixia Zhang. A Comparative Study of the DNS Design with DHT-Based Alternatives. In *Proceedings of the International Conference on Computer Communications (INFOCOM)*, pages 1–13, 2006. 106
- [117] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980. 12
- [118] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 129–140, 1991. 114

- [119] Andrzej Pelc and David Peleg. Broadcasting with Locally Bounded Byzantine Faults. *Information Processing Letters*, 93(3):109–115, 2005. 16, 17, 27, 50, 51, 52, 53, 71
- [120] Andrzej Pelc and David Peleg. Feasibility and complexity of broadcasting with random transmission failures. In *Proceedings of the 24th Symposium on Principles of Distributed Computing*, pages 334–341, 2005. 28
- [121] Konstantinos Pelechrinis, Marios Iliofotou, and Srikanth V. Krishnamurthy. Denial of Service Attacks in Wireless Networks: The Case of Jammers. *IEEE Communications Surveys & Tutorials*, 13(2):245–257, 2011. 9, 22
- [122] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003. 134
- [123] Krzysztof Piotrowski, Peter Langendoerfer, and Steffen Peter. How Public Key Cryptography Influences Wireless Sensor Node Lifetime. In *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks*, pages 169–176, 2006. 104
- [124] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, 2005. 38
- [125] Prolexic Technologies, Inc. www.prolexic.com. 29
- [126] Iyappan Ramachandran and Sumt Roy. Clear Channel Assessment in Energy-Constrained Wideband Wireless Networks. *IEEE Wireless Communications*, 14(3):70–78, 2007. 39
- [127] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, 2001. 3, 18

- [128] Michael K. Reiter. The Rampart Toolkit for Building High-Integrity Services. In *Proceedings of the International Workshop on Theory and Practice in Distributed Systems*, pages 99–110, 1995. 109
- [129] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiawicz. Handling Churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*, pages 127–140, 2004. 137, 138
- [130] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. A Jamming-Resistant MAC Protocol for Multi-Hop Wireless Networks. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, 2010. 28, 39, 45
- [131] Rodrigo Rodrigues, Petr Kouznetsov, and Bobby Bhattacharjee. Large-scale byzantine fault tolerance: Safe but not always live. In *Proceedings of the 3rd workshop on on Hot Topics in System Dependability*, 2007. 109, 110
- [132] Rodrigo Rodrigues and Barbara Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. Technical Report TR/932, MIT LCS, December 2003. 109
- [133] Rodrigo Rodrigues, Barbara Liskov, and Liuba Shrira. The Design of a Robust Peer-to-Peer System. In *Proceedings of the 10th ACM SIGOPS European Workshop*, page 2002, 117-124. 109
- [134] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001. 3
- [135] Jared Saia and Maxwell Young. Reducing Communication Costs in Robust Peer-to-Peer Networks. *Information Processing Letters*, 106(4):152–158, 2008. 3, 21, 99, 106, 108, 110, 111, 112, 124, 125
- [136] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, pages 314–329, 2002. 138, 139

- [137] Christian Scheideler. How to Spread Adversarial Nodes? Rotate! In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, 2005. 3
- [138] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990. 109
- [139] Vyas Sekar and Jacobus Van Der Merwe. LADS: Large-scale Automated DDoS Detection System. In *Proceedings of the USENIX ATC*, pages 171–184, 2006. 59
- [140] Micah Sherr, Michael Greenwald, Carl A. Gunter, Sanjeev Khanna, and Santosh S. Venkatesh. Mitigating DoS Attack Through Selective Bin Verification. In *Proceedings of the First international conference on Secure network protocols*, NPSEC’05, pages 7–12, 2005. 58
- [141] Victor Shoup. Practical Threshold Signatures. In *Advances in Cryptology - EUROCRYPT*, pages 207–220, 2000. 115
- [142] Emil Sit and Robert Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pages 261–269, 2002. 3, 106, 116
- [143] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental study of the effects of Transmission Power Control and Black-listing in Wireless Sensor Networks. In *Proceedings of the 1st IEEE Conference on Sensor and Adhoc Communication and Networks*, pages 289–298, Santa Clara, California, USA, October 2004. IEEE. 56
- [144] Dongjin Son, Bhaskar Krishnamachari, and John Heidemann. Experimental Study of Concurrent Transmission in Wireless Sensor Networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys, pages 237–250, 2006. 56
- [145] Kannan Srinivasan and Philip Levis. RSSI is Under Appreciated. In *Proceedings of the 3rd Workshop on Embedded Networked Sensors (Em-Nets)*, 2006. 39

- [146] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A Global View of KAD. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference*, pages 117 – 122, 2007. 1, 106, 138
- [147] Douglas R. Stinson. *Cryptography: Theory and Practice, 3rd Ed.* Chapman & Hall, 2006. 65
- [148] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM Conference on Applications, technologies, architectures, and Protocols for Computer Communications*, pages 149–160, 2001. 3, 18, 108
- [149] James Sundali and Rachel Croson. Biases in Casino Betting: The Hot Hand and the Gamblers Fallacy. *Judgment and Decision Making*, 1(1):1–12, 2006. 32
- [150] Vinod Vaikuntanathan. Brief announcement: Broadcast in Radio Networks in the Presence of Byzantine Adversaries. In *Proceedings of the 24th Annual ACM symposium on Principles of Distributed Computing*, pages 167–167, 2005. 13, 28
- [151] Ashlee Vance. WikiLeaks Struggles to Stay Online After Attacks. www.nytimes.com/2010/12/04/world/europe/04domain.html?_r=2&hp, 2010. 22
- [152] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS Defense by Offense. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 303–314, 2006. 29, 57, 58, 59
- [153] Dan Wallach. A Survey of Peer-to-Peer Security Issues. In *International Symposium on Software Security (ISSS)*, pages 42–57, 2002. 3, 106, 116
- [154] John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. *Security in Distributed, Grid, Mobile, and Pervasive Computing. Chapter 17: Wireless Sensor Network Security: A Survey.* Auerbach Publications, 2007. 3, 38, 39

- [155] Arvinderpal S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, and Sheueling Chang Shantz. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *Proceedings of the 3rd International Conference on Pervasive Computing and Communications*, pages 324–328, 2005. 104
- [156] Lan Wang and Yang Xiao. A Survey of Energy-Efficient Scheduling Mechanisms in Sensor Networks. *Mobile Networks and Applications*, 11:723–740, 2006. 61
- [157] Peng Wang, Nicholas Hopper, Ivan Osipkov, and Yongdae Kim. Myrmic: Secure and Robust DHT Routing. Technical Report 2006/20, University of Minnesota, 2006. 109
- [158] Qin Wang, Mark Hempstead, and Woodward Yang. A Realistic Power Consumption Model for Wireless Sensor Network Devices. In *Proceedings of the 3rd Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2006. 61
- [159] Yong Wang, Garhan Attebury, and Byrav Ramamurthy. A Survey of Security Issues in Wireless Sensor Networks. *IEEE Communications Surveys & Tutorials*, 8:2–23, 2006. 3
- [160] Mohamed Watfa, William Daher, and Hisham Al Azar. A Sensor Network Data Aggregation Technique. *International Journal of Computer Theory and Engineering*, 1(1):19–26, 2009. 105
- [161] Ronald Watro, Derrick Kong, Sue fen Cuti, Charles Gariner, Charles Lynn, and Peter Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 59–64, 2004. 39
- [162] Anthony D. Wood and John A. Stankovic. Denial of Service in Sensor Networks. *Computer*, 35(10):54–62, 2002. 28, 38, 39, 45
- [163] ALERT. www.alertsystems.org. 1, 3
- [164] Wenyuan Xu, Ke Ma, Wade Trappe, and Yanyong Zhang. Jamming Sensor Networks: Attack and Defense Strategies. *IEEE Network*, 20(3):41–47, 2006. 28

- [165] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 46–57, 2005. 28, 37
- [166] Andrew Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977. 163, 164, 165
- [167] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 21st International Conference on Computer Communications (INFOCOM)*, pages 1567–1576, 2002. 38
- [168] Maxwell Young and Raouf Boutaba. Overcoming Adversaries in Sensor Networks: A Survey of Theoretical Models and Algorithmic Approaches for Tolerating Malicious Interference. Accepted to *IEEE Communications Surveys & Tutorials*, 2011. 17, 28
- [169] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical Robust Communication in DHTs Tolerating a Byzantine Adversary. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 263–272, 2010. 3, 11, 134
- [170] Kaan Yüksel, Jens-Peter Kaps, and Berk Sunar. Universal Hash Functions for Emerging Ultra-Low-Power Networks. In *Proceedings of the Communications Networks and Distributed Systems Modeling and Simulation Conference (CNDS)*, 2004. 103, 104
- [171] Demetrios Zeinalipour-Yazti, Som Chandra Neema, Vana Kalogeraki, Dimitrios Gunopulos, and Walid Najjar. Data Acquisition in Sensor Networks with Large Memories. In *Proceedings of the 21st International Conference on Data Engineering Workshops*, pages 1188–1188, 2005. 103
- [172] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Area Location and Routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley Technical Report, April 2001. 3, 18

- [173] Marco Zuniga and Bhaskar Krishnamachari. Analyzing the Transitional Region in Low Power Wireless Links. In *Proceedings of the 1st IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, pages 517–526, 2004. 57

Appendix

The following two lemmas are relevant to the lower bounds in Chapter 4.

Lemma 15. $\Omega(\sqrt{n})$ expected queries are necessary in the single stream case.

Proof. In the following, let \tilde{O} denote that logarithmic factors are ignored. We follow Yao's min-max method [166] to prove lower bounds on any randomized algorithm that errs with probability no greater than $\lambda = 1/2^{\tilde{O}(\sqrt{n})}$: We describe an input distribution and show that any deterministic algorithm that errs with tolerance (average error) less than $2\lambda = 1/2^{\tilde{O}(\sqrt{n})}$ on this input distribution requires $\Omega(\sqrt{n})$ queries on average for this distribution. By [166], this implies that the complexity of any randomized algorithm with error λ has cost $(1/2)\Omega(\sqrt{n}) = \Omega(\sqrt{n})$. Let $[a, b]$ denote the bits in position $a, a + 1, \dots, b - 1, b$ of the stream. The distribution is as follows:

CASE 1. With probability $1/2$, \sqrt{n} uniformly distributed random bits in $[1, n/2]$ are set to 1 and the remaining bits in that interval are 0, $[n/2 + 1, n/2 + \sqrt{n}]$ are all set to 0, and the remaining bits are 1.

CASE 2.x: For $x = 0, \dots, \sqrt{n} - 1$, with probability $1/(2\sqrt{n})$, $[1, \dots, n/2]$ contains a uniformly distributed random set of x 0's and the rest are 1's; $[n/2 + 1, n/2 + \sqrt{n}]$ contains a uniformly distributed random set of x 1's and the rest are 0's; and the remaining bits in the stream are 0.

Analysis: Let A be a deterministic algorithm which errs with average probability less than 2λ . Note that A is completely specified by a list L of indices of bits to query while it has not yet discovered a 1, since it stops as soon as it sees a 1. Let x be the number of queries in the list that lie in $[1, n/2]$. For a constant fraction of inputs in CASE 1, A will not find a 1 in $[1, n/2]$ within \sqrt{n} queries. Hence either $x \geq \sqrt{n}$ or A must find a 1 with high probability in $[n/2 + 1, n]$. Now suppose $x < \sqrt{n}$. We show that A 's list L must

contain greater than $\sqrt{n} - x$ bit positions in $[n/2 + 1, n/2 + \sqrt{n}]$. To show this, assume this is untrue. Then A will err on the input in CASE 2.x in which all the x positions queried in $[1, n/2]$ and the $\sqrt{n} - x$ positions queried in $[n/2 + 1, n/2 + \sqrt{n}]$ are 0. Note that this input occurs with probability $(2\sqrt{n})^{-1} \binom{n/2}{x}^{-1} \binom{\sqrt{n}}{x}^{-1} \geq 2\lambda$ in the distribution. Therefore, the algorithm errs with probability at least 2λ ; this is a contradiction. We conclude that any algorithm erring with probability less than 2λ must either have $x \geq \sqrt{n}$ or queries greater than $\sqrt{n} - x$ bits of $[n/2 + 1, n/2 + \sqrt{n}]$.

Now we show that any such deterministic algorithm incurs an average cost of $\Omega(\sqrt{n})$ on the CASE 1 strings in this distribution. If $x \geq \sqrt{n}$ then for a constant fraction of strings in CASE 1, the algorithm will ask at least \sqrt{n} queries in $[1, n/2]$ without finding a 1. If $x < \sqrt{n}$, then with constant probability the algorithm will incur a cost of x in $[1, ..n/2]$ and go on to incur a cost of $\sqrt{n} - x$ in $[n/2 + 1, n/2 + \sqrt{n}]$ since all the values there are 0. Therefore, the distributional complexity with error 2λ is $\Omega(\sqrt{n})$. It follows from [166] that the randomized complexity with error λ is $\Omega(\sqrt{n})$. \square

Lemma 18. $\Omega(\log^{(i+2)} n)$ expected queries are required for a randomized algorithm that errs with probability less than $\lambda = (\ln^{(i)} n)^{-\epsilon}$ on one stream of length n . In particular, when $i = 0$, $\Omega(\log \log n)$ expected queries are required for a randomized algorithm with error less than $1/n^\epsilon$, for any constant $\epsilon > 0$.

Proof. We apply Yao's min-max method [166] and consider the distribution in which with probability $1/3$, one of the $I_1 = [1, n/3]$, $I_2 = [n/3 + 1, 2n/3]$, and $I_3 = [2n/3 + 1, n]$ intervals is all 0's, and the other two each contain exactly $n/4$ 1's with the 1's distributed uniformly at random. Let L denote the list of queries of a deterministic algorithm, and let x_i be the number of queries in $L \cap I_i$. The probability that the algorithm fails to find a 1 in any interval I_i is $\binom{n/3-x_i}{n/4} / \binom{n/3}{n/4} = \frac{n/12 - x_i}{n/3} \frac{n/12-1}{n/3-1} \cdots \frac{n/12-x_i+1}{n/3-x_i+1} > \left(\frac{n/12-x_i+1}{n/3-x_i+1}\right)^{x_i} > \left(\frac{1}{4} - \frac{3x_i}{n}\right)^{x_i} > \left(\frac{1}{4} - \epsilon\right)^{x_i} > \left(\frac{1}{e^{7/4}}\right)^{x_i} = e^{-7x_i/4}$ when $x_i = o(n)$ for sufficiently large n . Let I_i and I_j be the intervals that are not all 0's. Then the probability of failing to find a 1 in either I_i and I_j is $> e^{-7(x_i+x_j)/4}$ for sufficiently large n when $x_i + x_j = o(n)$. Hence the probability of not finding a 1 over all intervals is $> (1/3)e^{-7(x_i+x_j)/4} > 2\lambda$ if $x_i + x_j < (3/7)\epsilon \lg^{(i+1)} n$. We conclude that a deterministic algorithm with average error less than 2λ can have at most one $x_i, i = 1, 2, 3$ such that $x_i < (3/14)\epsilon \lg^{(i+1)} n$.

Now we examine the cost of such an algorithm. Suppose $x_1 \geq (3\epsilon/14) (\ln^{(i+2)} n)$ then with probability $1/3$ I_1 is all 0's and the cost incurred is x_1 , for an average cost of $(\epsilon/14)(\ln^{(i+2)} n)$. Now suppose $x_1 < (3\epsilon/14) \ln^{(i+2)} n$. From above, we know $x_2 > (3\epsilon/14) \ln^{(i+1)} n$. Then with probability $1/3$, I_2 is all 0's and with probability $> e^{-7x_1/4} > (\ln^{(i+1)} n)^{-3\epsilon/8}$, the algorithm does not find a 1 in I_1 and incurs a cost of $(3\epsilon/14) \lg^{(i+1)} n$ in I_2 for an average cost of at least $(\epsilon/14)(\ln^{(i+1)} n)^{1-3\epsilon/8}$. Hence the average cost of any such deterministic algorithm is at least $\min\{(\epsilon/14)(\ln^{(i+2)} n), (\epsilon/14)(\lg^{(i+1)} n)^{1-3\epsilon/8}\} = \Omega(\ln^{(i+2)} n)$. By Yao's min-max method [166], any randomized algorithm with error λ is bounded below by $1/2$ the average cost of a deterministic algorithm with average error 2λ on any distribution. The lemma now follows. \square