Securing Every Bit: Authenticated Broadcast in Radio Networks

Dan Alistarh* EPFL dan.alistarh@epfl.ch
Seth Gilbert EPFL seth.gilbert@epfl.ch
Rachid Guerraoui EPFL rachid.guerraoui@epfl.ch
Zarko Milosevic EPFL zarko.milosevic@epfl.ch
Calvin Newport MIT cnewport@csail.mit.edu

ABSTRACT

This paper studies non-cryptographic authenticated broadcast in radio networks subject to malicious failures. We introduce two protocols that address this problem. The first, NeighborWatchRB, makes use of a novel strategy in which honest devices monitor their neighbors for malicious behavior. Second, we present a more robust variant, MultiPathRB, that tolerates the maximum possible density of malicious devices per region, using an elaborate voting strategy. We also introduce a new proof technique to show that both protocols ensure asymptotically optimal running time.

We demonstrate the fault tolerance of our protocols through extensive simulation. Simulations show the practical superiority of the NeighborWatchRB protocol (an advantage hidden in the constants of the asymptotic complexity). The NeighborWatchRB protocol even performs relatively well when compared to the simple, fast epidemic protocols commonly used in the radio setting, protocols that tolerate no malicious faults. We therefore believe that the overhead for ensuring authenticated broadcast is reasonable, especially in applications that use authenticated broadcast only when necessary, such as distributing an authenticated digest.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: Fault Tolerance

General Terms

Algorithms, Security, Reliability

Keywords

Broadcast, Wireless networks, Byzantine faults

*This author's work was supported by the Swiss NCCR MICS project.

SPAA'10, June 13–15, 2010, Thira, Santorini, Greece.

1. INTRODUCTION

We study the problem of non-cryptographic authenticated broadcast in a multi-hop radio network. Consider a single source attempting to disseminate an important message to every device in an ad hoc wireless network. This broadcast should be *reliable*—every device receives the message—and *authenticated*—a device should only accept the message actually sent by the source. These properties should still hold even if some of the network devices are corrupted and behaving in an unpredictable, perhaps adversarial manner.

When public-key cryptography is available, this problem is solved with relative ease: the message is signed by the base station, and flooded through the network. Each receiver can verify the signature. Public-key cryptography, however, can be quite expensive, and is often impractical in resource-constrained settings such as sensor networks. Even lightweight cryptographic techniques, such as message authentication codes, may prove too costly in terms of computation or deployment costs. With this in mind, in this paper we study *non-cryptographic* authenticated broadcast.

Contribution. We present and analyze two protocols for authenticated reliable broadcast in multi-hop wireless networks subject to malicious failures. The MultiPathRB protocol is the first protocol that is provably optimally resilient and achieves asymptotically optimal running time. The NeighborWatchRB protocol, by contrast, is less robust in theory, but performs much better in practice, as seen in our simulations.

System Overview. The key challenge we face is that some of the devices in the network may be corrupted. Formally, we capture this corruption with *Byzantine* failures. Devices suffering from Byzantine faults can disrupt a protocol in several ways: they may remain silent when required to broadcast; they may lie about their view of the system; they may spoof messages from other devices; they may overpower honest messages, replacing them with dishonest messages; or they may jam the airwaves with noise, preventing any communication. In placing no limitations on the behavior of Byzantine devices, we capture scenarios where devices are simply *malfunctioning*, as well as scenarios where devices have been compromised by a malicious hacker. An advantage of this approach is that the resulting protocols are resilient to *all possible* attacks, rather than focusing on a particular known attack. A protocol proved correct is this setting provides a strong measure of comfort to the practitioner deploying it in an unpredictable real world setting. As established by Koo [22], reliable broadcast is impossible

if more than 1/4 of a device's neighbors are Byzantine. We thus assume that the density of malicious devices does not exceed this bound.

Wireless devices in the network have a few capabilities that help to overcome this challenging environment. First, they can perform *carrier sensing* in order to determine whether or not the channel is currently in use (this is also referred to as *collision detection* in the literature). That is, if there is some activity on the channel—be it a single message being sent, a collision of multiple messages, or a malicious device jamming the airwaves—the protocol can distinguish this case from the case of no activity. (Implementation details for such a carrier-sensing MAC layer are discussed in [12].) Second, we also assume that each device has access to a localization service that provides an (approximate) location. (Such a service might calculate location directly, as in GPS or Cricket [30], or indirectly via trilateration, as in [29, 13]).

Metrics. We focus on two metrics: fault tolerance and running time. Our protocols aim to tolerate the maximum possible density of Byzantine faults in the network, and achieve an optimal running time as expressed in terms of the adversary's power. In more detail, if corrupt devices jam the channel in every round, then no protocol will ever complete. Continual disruption, however, is not sustainable in practice: it drains the batteries of the malicious devices, and it makes them easier to detect and eliminate. Thus, we analyze executions in which there is some bound β on the total number of messages broadcast by Byzantine devices in each neighborhood. It is easy to see that no protocol can terminate in less than $\Omega(\beta D)$ time, where D is the diameter of the network, since the adversary may jam transmission at every hop. And in [19], it was shown that no protocol can terminate in less than $\Omega(\log |\Sigma|)$ time, where Σ is the set of all possible messages. Our protocols match this combined lower bound of $\Omega(\beta D + \log |\Sigma|)$ rounds.

For this analysis, we consider a specific topology where devices are deployed at unit intervals in the plane, forming a grid (or mesh). We assume that each device can communicate with every other device that is within R units on the vertical axis and R units on the horizontal axis. We call this set of nodes within broadcast range the device's *neighborhood*. Let t be the maximum number of Byzantine nodes in a neighborhood. (This model has been previously used in, e.g., [22, 4, 23].) Extension of the analytic bounds to arbitrary topologies is left as important future work.

The NeighborWatchRB Protocol. Our first main result in this paper is a protocol, NeighborWatchRB, which tolerates malicious interference while guaranteeing an asymptotically optimal running time of $O(\beta D + \log |\Sigma|)$. There are two key challenges in devising such a protocol. First, devices cannot necessarily determine which device sent a given message; that is, Byzantine devices may effectively spoof messages. Second, in order to reach the edge of the network, the message from the source is passed along a multi-hop path; the receiver must ensure that it has not been modified along the way by a malicious user. Our protocol is divided into two levels to cope with these different problems.

Single hop level: At the lower level, we develop a subprotocol that can send a block of bits, i.e. a multi-bit message, over a single hop of a multi-hop network. The key idea is to use silent rounds of communication to confirm that the transmission has been successful. An additional challenge in the multi-hop setting is to synchronize the sender and the receiver, ensuring that they agree on the order of the transmitted bits. Some ideas are common to the one-hop protocol of [19, 20]; however, new insights are required for tolerating adversaries in the multi-hop setting.

Multi-hop level: Next, we implement a complete multi-hop broadcast protocol on top of the previous layer. The idea is that honest devices are clustered into regions, and they implement a "neighborhood watch" system by actively preventing devices in their region from disseminating corrupted information.

The protocol is successful as long as there is no region occupied solely by Byzantine devices; this requirement translates into the bound $t < \frac{1}{4}R^2$. We also introduce a "2-voting" variant that tolerates $t < \frac{1}{2}R^2$. The protocol is *adaptive*, in that the message is delivered as soon as Byzantine interference stops. Unlike the protocols of [23, 5], nodes do not need to know the bound β on the Byzantine budget; moreover, they do not even need to know the bound t on the number of Byzantine nodes in their area.

The MultiPathRB Variant. Second, we introduce a variant of the protocol that achieves optimal fault tolerance, i.e., it tolerates $t < \frac{1}{2}R(2R+1)$, while maintaining asymptotically optimal running time. The protocol re-uses level one of NeighborWatchRB, and employs a voting strategy to ensure multi-hop authentication. A similar strategy is used by the protocol of [5]. However, note that this previous protocol assumed authenticated communication, and did not allow Byzantine nodes to jam the network or otherwise disrupt communication. Further, our variant improves on the protocol of [5] by adapting to the *actual* number of malicious broadcasts, by delivering the message as soon as interference stops.

Analysis. An important technical contribution of this paper is the upper bound on the running time of the two protocols. (Prior work or reliable broadcast, e.g. [22], focused on feasibility, omitting any analysis of performance.) First, note that combining the single-hop transmission layer and the multi-hop layer in the natural way results in protocols with sub-optimal running time of $\Omega(\beta \cdot D \cdot \log |\Sigma|)$, since the adversary may jam the message for $\Omega(\beta)$ rounds at each network hop. The protocols we introduce carefully integrate the sub-protocols to achieve a *pipelined* data flow, and hence the optimal running time of $O(\beta \cdot D + \log |\Sigma|)$. The main technical difficulty in the analysis is showing that the pipeline continues to flow, and cannot be (too) disrupted by interference (see Theorem 5).

Evaluation. We evaluate both protocols using the WSNet wireless network simulator (Section 6). Devices are deployed at random in a two-dimensional plane, using both uniform and clustered distributions. We focus on three different models of faulty behavior: crash failures, jamming devices that aim to delay the protocol, and malicious attacks (i.e., Byzantine devices that attempt to spread a fake message).

In all three cases, both protocols show good levels of fault tolerance. In general, we observe a close link between the density of deployment and the number of Byzantine devices that the protocol can tolerate. We also notice that, in practice, if the bad devices are distributed at random, then the supposedly less robust NeighborWatchRB protocol easily outperforms MultiPathRB, even for greater numbers of bad devices.

We also compare the performance of the NeighborWatchRB

protocol to a simple epidemic flooding protocol. The latter protocol has no built-in fault tolerance, and can be disrupted by any Byzantine interference. We found that the NeighborWatchRB protocol takes about seven times longer to complete. In many ways, this shows the success of our pipelining strategy, as much of the additional single-hop cost is defrayed by the inherent cost of propagating a message across multiple hops.

Interpretation. One conclusion from these experiments is that, while there is some inherent overhead in tolerating Byzantine failures, it may be possible to engineer protocols that are quite efficient. For example, consider the dual-mode protocol that operates as follows: (a) every message is broadcast via simple epidemic flooding, using no security; and (b) a small digest of each message is broadcast using a protocol such as NeighborWatchRB. Good security is ensured as long as the digest is chosen appropriately. And as long as the digest is no more than 1/7 the size of the original message, the induced overhead may be tolerable. We conjecture, in fact, that the overhead in such an implementation may compare well to systems the rely on cryptography, while at the same time avoiding the costs of a cryptographic solution.

Because of space limitations, some proofs and figures are deferred to the full version of the paper, available at [18], together with the source code for the simulations.

2. RELATED WORK

Jamming in wireless networks is a problem that has been extensively studied by the applied networking community. The literature analyzes efficient jamming attacks (e.g., [26]), as well as mechanisms to detect and circumvent such attacks (e.g., [15, 14]). Recent work [3] shows both in theory and experimentally that adaptive jammers can efficiently reduce the throughput of a wireless network. Most defense mechanisms focus on preventing a specific attack, and are based on physical layer methods (e.g., [24, 25]) and on MAC layer strategies [2, 31].

Koo [22] studied the problem of broadcast in *multi-hop* wireless networks, under the assumption that single-hop communication is reliable and authenticated (i.e., no jamming or spoofing). He proved that no algorithm can tolerate more than $\frac{1}{2}R(2R+1)$ malicious devices per neighborhood. Bhandari and Vaidya [4, 5] present a protocol that matches this lower bound. The problem of broadcast in the context of malicious interference and jamming was also considered in [23, 19, 20], under weaker adversarial models than the one of this paper.

Another approach can be found in [11], where they develop "Integrity Codes" that allow the transfer of information in a single-hop network based on collision detection (i.e., carrier sensing). This premise, together with the coding technique, originates from Unidirectional Error-Detecting Codes [7, 10], and is similar to the assumptions we make in this paper. In [19, 20] the authors develop a broadcast protocol for single-hop wireless networks in the presence of Byzantine nodes. In this paper, we generalize their protocol in the case of multi-hop networks. Although the algorithm in [19, 20] is similar to our 1Hop-Protocol, new insights are required in order to tolerate Byzantine nodes in the multi-hop case.

Other lines of research have considered the power of an adversary that is restricted to behave in a probabilistic manner (e.g., [27, 6]), or solutions to the problem when more than one communication channel is available, via frequency hopping (e.g., [8], [16]) .

A parallel approach studies broadcast authentication using cryptographic techniques (e.g., [28, 1]). Boneh et al. [9] prove that authentication is impossible without relying on digital signatures or on synchronization. Perhaps the best known protocol in this line of research is μTESLA, by Perrig et al. [28], which relies on weak time synchronization between sender and receivers and on message authentication codes. A survey of recent work on lightweight cryptographic techniques is presented in [17].

3. MODEL

We now describe the two models of wireless networks that we rely on in this paper. The first is somewhat simplified, and is used for analysis; the second is more realistic, and is used for simulations.

Analytical model. For the purpose of analysis, we model a system consisting of devices, which we refer to as *nodes*, that communicate via wireless radio. Let R be the communication radius. We define a *neighborhood* of a node v to be the area within distance R of v. Additionally, each node knows its location.

Time is divided into slots, which we refer to as *rounds*, and we assume that slots are *small*; each round is large enough to transmit at most a few bits of data. Only one node in each neighborhood can send a message in each time slot without causing a collision. When two neighbors of a node v broadcast in the same round, v may receive either of the two messages, or no message at all. In the latter case v detects a *collision*.

Nodes are either *honest*, in which case they follow the protocol, or *Byzantine*, in which case they may deviate arbitrarily. Let t be the maximum number of Byzantine nodes in any neighborhood. Also, let β (the *budget*) be the maximum number of broadcasts for Byzantine nodes in a neighborhood.

While our algorithms remain correct for any network topology (as long as t is sufficiently low), we analyze performance in the context of a specific topology: a two-dimensional grid where nodes are placed at every grid point. We express our results in the L_∞ norm, meaning that, given two nodes $v = (x_1, y_1)$ and $w = (x_2, y_2)$, we say that v is in the neighborhood of w if $|x_2 - x_1| \leq R$ and $|y_2 - y_1| \leq R$. A similar model is used for analysis in [22, 23, 5].

Simulation model. Our simulations were performed using the WSNet Worldsens simulator [32]. The simulator uses the *Friis freespace* propagation model for radio wave propagation. Communication depends on the actual topology, as opposed to the analytical model. We modified the MAC layer to implement carrier sensing, i.e. to provide a notification of when the channel seems to be in use, but no message is delivered. The intent here was to emulate MAC layers that can detect when there are significant changes in the level of signal on the channel. The setup captures realistic behavior missed by our theoretical analysis (real topology, lost messages, capture effect). Simulated devices have access to a synchronized clock, and we use this clock to implement a fixed (TDMA-like) schedule. We discuss the implementation details in Section 6.

Relation between models. Although the theoretical analysis is performed in the L_∞ topology, the protocols remain provably correct in the actual geometry as well, as

long as the assumptions on the density of malicious nodes still hold. However, our experiments do not specifically enforce these assumptions (nodes are randomly distributed, an arbitrary fraction becomes malicious, packets may be lost, etc.), since the intention is to investigate the behavior of the protocols in a more realistic setting.

4. AUTHENTICATED BROADCAST

In this section, we present our authenticated broadcast protocols, sketching some of the key properties.

Notice that it is difficult for a node to trust the contents of a message, as it cannot determine whether that message originated at the ostensible sender s, or at a malicious device spoofing messages that claim to be from s. However, if a receiver does *not* receive a message in a round, it can be certain that no message was sent; the malicious nodes cannot "forge" silence. Thus, we make frequent use of silence to authenticate data.

To minimize the damage caused by a malicious broadcast, the message is transmitted and authenticated one bit at a time. In this way, we ensure that any interference by a malicious device can cause only a small amount of damage. Otherwise, a single malicious broadcast might force the entire protocol to restart!

Schedule. To prevent contention among honest nodes, we allocate a simple (TDMA-like) broadcast schedule such that no two nodes within distance $3R$ of each other are scheduled in the same round (recall that, in the analytical model, nodes are placed on a two-dimensional grid of unit length 1). Since each node knows its location, we can easily construct such a schedule *locally*, i.e. without any communication between nodes, by assigning each grid point a schedule slot, and re-using schedule slots as long as no two (honest) neighbors of a node may collide. It is straightforward to build such a schedule of size $O(R^2)$. At the beginning of the protocol, each node locally computes its schedule slot, and the schedule slots of its neighbors. For simplicity, each schedule slot is 6 consecutive rounds long, which we also call the *broadcast interval* of the node. During its interval, the node broadcasts messages, and may receive acknowledgements from its neighbors.

In the following, we divide the protocol into two layers, a lower layer responsible for single-hop propagation, and a higher layer responsible for end-to-end delivery.

Level 1: Single Hop Transmission

We split the presentation of the lower communication layer into two protocols: the 2Bit-Protocol, which transmits two bits across a single communication hop, and the 1Hop-Protocol, which ensures the transmission of a stream of bits over one hop.

Assume that an honest sender s attempts to transmit two bits $\langle b_1, b_2\rangle$ to a set of honest nodes P in the neighborhood of s. In the 2Bit-Protocol, the sender transmits the two bits via a pattern of broadcasts and silence: when it wants to send a '1', it broadcasts a message; when it wants to send a '0', it remains silent[1]. The receivers acknowledge the information, and two veto rounds are used to determine whether the protocol has succeeded. The protocol proceeds in six rounds (i.e., during one schedule slot):

R1) *Sender:* If $b_1 = 1$, then the sender s transmits a message '*bit1*'; otherwise, it remains silent.

R2) *Acknowledge:* Every receiver in P (the neighborhood of s) that receives a message or detects a collision in round R1, broadcasts a (non-empty) *bit1-response* message in round R2; otherwise, it remains silent. Notice that if $b_1 = 1$, then this acknowledgment is likely to cause a collision. If a node in P broadcasts a *bit1-response*, then it assumes that $b_1 = 1$; otherwise, it assumes that $b_1 = 0$.

R3) *Sender:* If $b_2 = 1$, then the sender transmits a *bit2* message; otherwise, it remains silent.

R4) *Acknowledge:* Every receiver in P that receives a message or detects a collision in round R3 broadcasts a *bit2-response* message in round R4; otherwise, it remains silent. If a node in P broadcasts a *bit2-response*, then it assumes that $b_2 = 1$; otherwise, it assumes that $b_2 = 0$.

R5) *Sender Veto:* The sender s transmits a veto message in any of the following cases:

- Bit $b_1 = 0$ and it receives a message or detects a collision in round R2.
- Bit $b_1 = 1$ and it does *not* receive a message or detect a collision in round R2.
- Bit $b_2 = 0$ and it receives a message or detect a collision in round R4.
- Bit $b_2 = 1$ and it does *not* receive a message or detects a collision in round R4.

R6) *Receiver veto:* Every node in P broadcasts a veto message if it receives a message or detects a collision in round R5.

If the sender s receives no messages and detects no collisions in round R6, then it returns success; otherwise, it returns failure. Similarly, if a receiver in P receives no messages and detects no collision in round R5, then it returns success along with its estimate of b_1 and b_2; otherwise, it returns failure.The following theorem captures the key properties of this sub-protocol:

THEOREM 1. *If honest sender v and honest receivers P in the neighborhood of v begin the* 2Bit-Protocol *in the same round. Then at the end of the sixth round:*
• Authenticity: *A receiver returns bits $\langle b_1, b_2\rangle$ only if the sender s sent $\langle b_1, b_2\rangle$.*
• Termination: *Sender v returns* success *only if every honest node in v's neighborhood returns* success.
• Energy: *If sender or receiver returns* failure*, then a Byzantine device in the neighborhood of s expended at least one broadcast.*

We continue with the 1Hop-Protocol, in which a sender s reliably sends a constant-sized message to a set of honest receivers P in its neighborhood. Unlike the 2Bit-Protocol, which can fail, the 1Hop-Protocol always delivers the message eventually. The 1Hop-Protocol divides the message into

[1]Notice, of course, that a physical layer implementation will most likely encode both a "broadcast" and "silence" via some waveforms; the key property is that the Byzantine nodes cannot overwrite a broadcast with silence. We continue using the terminology of "broadcasts" and "silence," leaving physical layer implementations for future work.

individual bits, and each bit, along with some control information, is sent using the 2Bit-Protocol. Whenever the 2Bit-Protocol returns failure, the failed bit is re-transmitted.

The key difficulty involves synchronizing the sender and the receivers. If the sender successfully completes the protocol, then we know that the receivers delivered the bits. The converse, however does not hold: some (or all) of the receivers may finish receiving a bit, while the sender continues to repeat the transmission. The receivers should not confuse this repeated transmission with the next bit in the sequence.

To remedy this problem, we use an alternating bit strategy. That is, prior to sending each bit of the message, we send an additional control bit; this control bit alternates between '1' and '0'. Thus, the two bits sent by the 2Bit-Protocol are this alternating "parity" bit, and the one bit of data. The receiver can determine when the sender has advanced to a new bit by examining the parity bit. Note that the parity bit mechanism also ensures that silence on the sender side is not misinterpreted as a $\langle 0, 0\rangle$ transmission (the first value of the parity bit is '1'). The receiver delivers the entire message once it has received all the bits. The resulting protocol has the following properties:

THEOREM 2. *If honest sender v and honest receivers P in the neighborhood of v begin the* 1Hop-Protocol *in the same round:*
- Authenticity: *A receiver returns m only if v sent m.*
- Termination: *When the sender terminates, every node in P has received the message.*
- Energy: *If m consists of k bits, and v does not complete sending m for r rounds, then Byzantine nodes in the neighborhood of v expend at least $(r - 6k)/6$ broadcasts.*

Notice that the protocol requires $6k$ rounds to transmit the message in the absence of malicious interference. For the rest of $(r - 6k)$ rounds, the adversary needs to broadcast at least once every 6 rounds to prevent the delivery of the message. The claim regarding the energy usage by the Byzantine nodes follows because each bit is sent individually; otherwise, a single disruption by the adversary would force the entire message to be repeated.

Level 2: NeighborWatchRB

NeighborWatchRB implements authenticated broadcast across multiple hops on top of the 1Hop-Protocol. We partition the plane into squares of maximum size such that any two nodes located in neighboring squares are able to communicate. In the analytic model, the squares are of size $\lceil R/2\rceil \times \lceil R/2\rceil$, that is, each square contains $\lceil R/2\rceil^2$ nodes.

All the nodes in the same square act identically: whenever one broadcasts, they all broadcast; whenever one is silent, they all are silent. The clustering into squares and the assignment of schedule slots are performed without any communication, since each node knows its location. For example, if the 1Hop-Protocol calls for a silent round (to confirm the authenticity of a message), all the honest nodes in the square are silent; if the 1Hop-Protocol calls for a broadcast during a veto round, all the honest nodes broadcast (ensuring that the message is vetoed, as long as there is at least one honest node in the square). Effectively, all the nodes in a square act like one "meta-node."

In this case, each square is assigned to a schedule slot; as before, the schedule is chosen to avoid collisions between neighboring squares. Whenever a square is scheduled, all the nodes in that square execute the next step of the 1Hop-Protocol. The source node is the only exception: it behaves independently of any square and it always is awarded the first broadcast interval in the schedule. The source begins the protocol by using the 1Hop-Protocol to disseminate the bits of the message to its neighbors.

Each node maintains a buffer of bits received through the 1Hop-Protocol for each of its neighboring squares (plus the source, for nodes that are in range of the source). We say that a node *commits* to bit number i if it has received bits number $1, 2, \ldots, i$ from one of its neighbors, through the 1Hop-Protocol.

Once a node has committed to a new bit, it executes rounds of the 1Hop-Protocol for that bit whenever its square is scheduled, until the 1Hop-Protocol returns success. Note that a node will not proceed to broadcasting bit number $i+1$ as long as bit number i has not been successfully broadcast.

Assume node n has no new bits to send. However, it notices that a node is trying to broadcast a new bit during n's broadcast interval. Then node n blocks the 1Hop-Protocol initiated by the other node, by broadcasting during veto rounds. Thus, we can be sure that data is propagated only when every node in a square has committed to it. Note that, in this way, the protocol will perform correctly if there is no Byzantine interference.

A Byzantine node may disrupt the protocol either by trying to disseminate corrupt information, or by broadcasting during veto rounds. However, in order to delay the broadcast, the Byzantine node must continue to expend its broadcast budget. On the other hand, notice that a Byzantine node cannot relay bad data, as it can only propagate a bit when every node in the square agrees on that bit. Thus, as long as there is at least one honest node in every square of size $\lceil R/2\rceil \times \lceil R/2\rceil$, that is $t < \lceil R/2\rceil^2$, the protocol succeeds.

An alternate way of viewing the protocol is as creating a grid of *honest* meta-nodes located at the center of each $\lceil R/2\rceil \times \lceil R/2\rceil$ square, where any two neighboring meta-nodes can communicate. If a device tries to deviate from the behavior of its corresponding "meta-node," then it is vetoed by honest members of the cluster. The spreading of the message implements a simple epidemic protocol on top of this grid. To gain efficiency, one could use a more complex routing protocol on this overlay. The following theorem guarantees correctness of the protocol.

THEOREM 3. *If an honest source sends m and if $t < \lceil R/2\rceil^2$, then: (1) If a node commits to bit b_i, then b_i is the i^{th} bit of m. (2) Eventually every node commits to every bit of the message.*

We also consider a variant of NeighborWatchRB which we refer to as "2-voting" NeighborWatchRB: in this case, a node only commits to a bit if it receives it from two different neighboring squares. Thus, the protocol is able to tolerate roughly $t < R^2/2$ adversaries per neighborhood.

Level 2: MultiPathRB

We conclude by describing a variant of the top layer which implements the MultiPathRB protocol. MultiPathRB also uses the 1Hop-Protocol for single hop authentication. Instead of grouping devices into meta-nodes, it employs an elaborate voting strategy at the multi-hop level in order to tolerate the maximal number of adversaries. More precisely,

in order to deliver a message, it must be transmitted along a sufficient number of node-disjoint paths located in the same neighborhood. This strategy was first introduced in [5].

Specifically, the protocol uses three types of messages: SOURCE messages, COMMIT messages, and HEARD messages. Initially, for each bit b_i of the message, the source s sends a message $\langle \mathsf{SOURCE}, b_i \rangle$ using the 1Hop-Protocol. Every neighbor of s can commit to any bit that it receives directly from the source, as Theorem 2 guarantees the authenticity of the message.

When a node commits to a bit b_i, it sends a $\langle \mathsf{COMMIT}, b_i \rangle$ message using the 1Hop-Protocol. When a node receives $\langle \mathsf{COMMIT}, b_i \rangle$ from some node v, it sends a $\langle \mathsf{HEARD}, v, b_i \rangle$ message. We say that v is the *cause* of the HEARD message.

A node can commit to a bit when it has received at least $t+1$ COMMIT and HEARD messages, such that: there is some neighborhood N where (a) the source of every COMMIT message, (b) the source of every HEARD message, and (c) the cause of every HEARD message all lie in that neighborhood N. Recall that a node identifies the location of a message's sender based on the slot in the broadcast schedule in which the message has been sent. Since at most t of these nodes are dishonest, at least one version of the received message must be correct. This implies that the broadcast protocol never delivers a fake message (authentication). Following the analysis in [5], we can also show that the protocol always terminates.

THEOREM 4. *If an honest source sends m via* MultiPathRB*, and if $t < \frac{1}{2}R(2R+1)$, then (1) a node commits to b_i only if b_i is the i^{th} bit of m and (2) eventually every node commits to every bit of m.*

5. RUNNING TIME ANALYSIS

In this section, we analyze the performance of MultiPathRB. The upper bound on the running time of NeighborWatchRB will follow as a special case.

There are two issues that arise. The first issue is related to bounding adversarial interference as the data traverses the network: at each hop, the Byzantine nodes can delay different parts of the protocol. The second issue is related to bounding queuing delays at the nodes themselves: a given node may have a long queue of protocol messages to send (e.g., COMMIT messages, HEARD messages, etc.). The adversary may cause more delay than expected by creating a backlog of messages, thus initiating a data "traffic jam." The argument is structured to show that such a jam does not affect the asymptotic running time.

Proof Preliminaries. Each SOURCE, COMMIT and HEARD message is of size $O(1)$, consisting of an identifier indicating its type, along with the value of the transmitted bit; the HEARD message also includes the identifier of the node that caused the HEARD message—the identifier can be encoded in $O(\log R)$ bits by its relative location from the sender. Treating R as constant (as we do throughout), each message consists of only $O(1)$ bits.

Recall that each node is scheduled at least once every $O(R^2)$ slots. As a corollary of Theorem 2, we conclude that if the adversary uses $\leq \beta_i$ broadcasts to delay a particular SOURCE, COMMIT, or HEARD message, then the message transmission completes in $O(\beta_i)$ scheduled slots, i.e., in $O(\beta_i)$ time.

For each node q, our goal is to bound the time it takes for q to receive the message from the source. We first isolate a rectilinear path from the source s to q that is of width $2R+1$ and extends distance R beyond q (see Figure 1). For each bit i, we can trace the progression of messages along this path. We say that a node is in layer k if a path to the source takes no more than k_v vertical hops, k_h horizontal hops, and either ($k = k_v$ and $k_h = 0$) or ($k = k_h + k_v + 2R$ and $k_h \neq 0$). See Figure 1 for a layer numbering example.

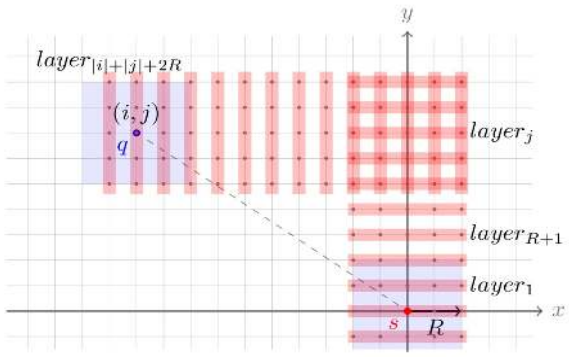

Figure 1: Rectilinear path of width $2R+1$ from source to node q.

For bit i, we introduce the following notation:

- Let S_i be the time interval during which the source transmits its *SOURCE* message for bit number i.
- Let $C^*_{i,[1,R]}$ be the interval during which nodes in layers $[1, R]$ transmit *COMMIT* messages associated with bit i.
- Let $C_{i,k}$ be the interval during which nodes in layer k transmit *COMMIT* messages for bit i.
- Let H^*_i be the interval during which the nodes in layers $[k+1, k+R]$ send *HEARD* messages caused by COMMIT messages for bit i from nodes in layers $[1, R]$.
- Let $H_{i,[k+1,k+R]}$ be the interval during which the nodes in layers $[k+1, k+R]$ send *HEARD* messages caused by COMMIT messages for bit i from nodes in layer k.

A sender transmits a COMMIT message for bit i before transmitting a COMMIT message for bit $i+1$. Also, a sender transmits its HEARD messages for bit i before transmitting HEARD messages for bit number $i+1$.

Thus we can split the execution into segments, as depicted in Figure 2. Each row in Figure 2 follows an individual bit as it progresses through the path, while the columns capture the events that may occur concurrently. Formally, a segment G_i is the smallest interval that contains all the intervals in column i.

Different segments may overlap. However, given two segments G_i and G_j, where $i > j$, the protocol ensures that G_i finishes *after* G_j finishes. Moreover, G_{i+1} begins, at the latest, as soon as G_i completes. Thus, the running time can be bounded by the sum of the segment lengths.

Also, notice that by construction, a given node is involved, for each segment, in transmitting in at most $R+1$ different rows: one for a COMMIT message, and R for HEARD messages.

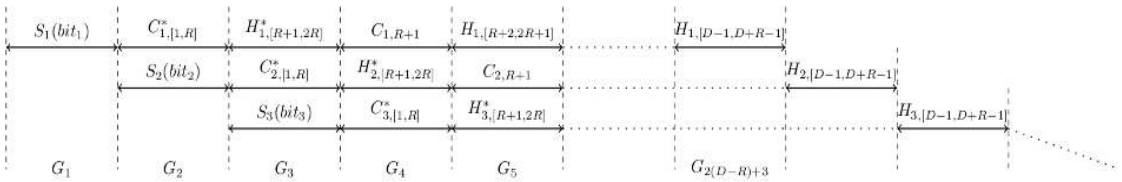

Figure 2: Execution segmentation for analyzing performance. Each vertical segment $\langle G_1, G_2, \ldots \rangle$ captures a set of events that occur concurrently. Each horizontal row traces a single bit along the path from source to target.

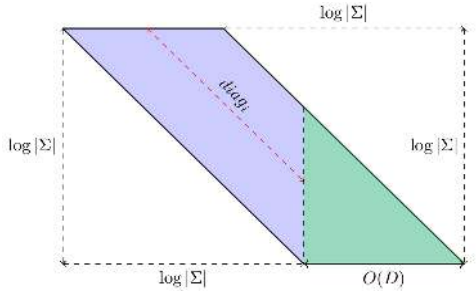

Figure 3: When $D \leq \log|\Sigma|$, we analyze the first $\log|\Sigma|$ segments separately from the remaining $O(D)$ segments. In the first $\log|\Sigma|$ segments, there are at most $O(D)$ diagonal stripes, each of which captures the behavior of a single layer, and hence can be delayed by at most $O(\beta)$ adversarial broadcasts.

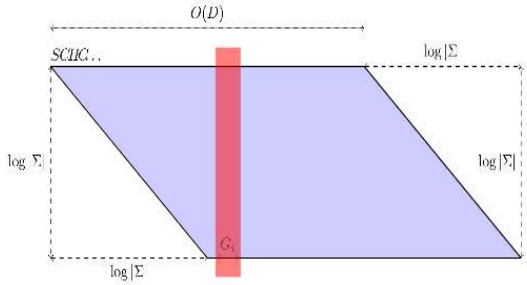

Figure 4: When $D > \log|\Sigma|$, there are $O(D)$ segments. Each node may have to broadcast at most $O(R^3)$ messages in a segment. It follows that a single node may be delayed for at most $O(R^3\beta)$ rounds within a segment. Since there are $O(D)$ segments in this case, we obtain that the total length of the execution is $O(\beta D)$.

Finally, observe that there are $\Theta(D) + \log|\Sigma|$ segments: $\Theta(D)$ segments for the first bit to travel across the path, and then $\log|\Sigma|$ segments while the remaining bits exit the pipelined path. The goal of the remainder of the section is to bound the number of rounds required to complete each of the segments.

Case 1: Large Diameter. Assume that the diameter of the network is large when compared to the message, specifically $D > \log|\Sigma|$ (see Figure 4).

As already indicated, a node p can appear in at most $R+1$ rows in a segment. For a given segment, for a given row, node p may have to broadcast $O(R^2)$ messages: if node p is at distance $> R$ from the source, it may have to broadcast $O(R^2)$ HEARD messages in a single segment. Thus, node p broadcasts at most $O(R^3)$ messages in a segment.

Notice that only Byzantine nodes situated at distance at most $2R$ from p can cause p's messages to be delayed. Thus, the total number of adversarial broadcasts that delay p is bounded by 4β. Since, as we observed above, β_i broadcasts can only delay a message for $O(\beta_i)$ time, we conclude that node p can broadcast all of its messages for a given segment in time $O(R^3\beta) = O(\beta)$. Finally, since $D > \log|\Sigma|$, there are $O(D)$ segments, and hence the total length of the execution is $O(\beta \cdot D)$.

Case 2: Small Diameter. Assume now that the diameter of the network is small, i.e., $D < \log|\Sigma|$, as is depicted in Figure 3. In this case, we focus on the first $\log|\Sigma|$ segments; there are at most $O(D)$ segments in the remainder, and, by the rationale above, their total length is bounded by $O(\beta \cdot D)$. Again, we are interested in determining the maximum delay for each segment. For segment G_i, let r_i be the row with the longest interval. If we assume that the adversary delays r_i with β_i broadcasts, we can bound the total delay of the first $\log|\Sigma|$ segments by $\sum_{i=1}^{\log|\Sigma|} O(1+\beta_i) = O\left(\log|\Sigma| + \sum_{i=1}^{\log|\Sigma|} \beta_i\right)$.

The segment tableau (see Figure 3) can be considered in terms of diagonal stripes from the top-left to the bottom-right. Notice that each such stripe involves nodes that are in the same layer along the path. Nodes in the same layer can be delayed by at most 9β broadcasts. Since there are at most $O(D)$ stripes, we can bound $\sum_{i=1}^{\log|\Sigma|} \beta_i$ by $O(\beta D)$.

We can obtain an asymptotic bound on the worst-case running time of NeighborWatchRB as a special case of the previous argument. Specifically, we transform MultiPathRB into NeighborWatchRB as follows: SOURCE and COMMIT messages are shortened to contain only the message bit;and HEARD messages are suppressed. Thus, we re-arrive at the same asymptotic bound of $O(\beta \cdot D + \log|\Sigma|)$ rounds. Thus we conclude:

THEOREM 5. *If an honest source sends a message m via* MultiPathRB *or via* NeighborWatchRB*, then every honest node delivers a message within time $O(\beta \cdot D + \log|\Sigma|)$.*

6. SIMULATION RESULTS

In this section, we simulate the two protocols presented in Section 4. Additional graphs and measurements can be found in the full version [18], together with the code required to run the simulations.

Methodology. Experiments were performed on maps of size varying from 20×20 to 60×60 length units with up

to 4000 nodes distributed either uniformly at random (in most simulations), or in a clustered fashion (where noted). We define the density as the total number of nodes divided by the area of the map. For most experiments, each node had an average broadcast range R of approximate 4 length units, and thus the network was between 7 and 21 hops from corner to corner. For an overview of the simulation model, see Section 3.

We implemented the two algorithms following the descriptions in Section 4. For the NeighborWatchRB protocol, the implementation assumes a (reduced) square size of $R/3 \times R/3$, in order to ensure propagation of messages between any two adjacent squares. We tested two variants of MultiPathRB with $t = 3$ and $t = 5$. (Here, t refers to the number of faults per region that the algorithm is tuned to tolerate.)

In each of our simulations, a single honest source node, located at the center of the network, initiates a broadcast of a short message. (Most experiments simulated the broadcast of a 4-bit message; longer messages simply increase the simulation times while yielding little additional insight.) Each experiment was repeated between 6 and 12 times, with outliers being discarded. We measured four parameters: how long the broadcast took to terminate, the percentage of nodes that completed the protocol, the number of broadcasts needed for all nodes to complete the protocol, and the percentage of completed nodes that received the correct message.

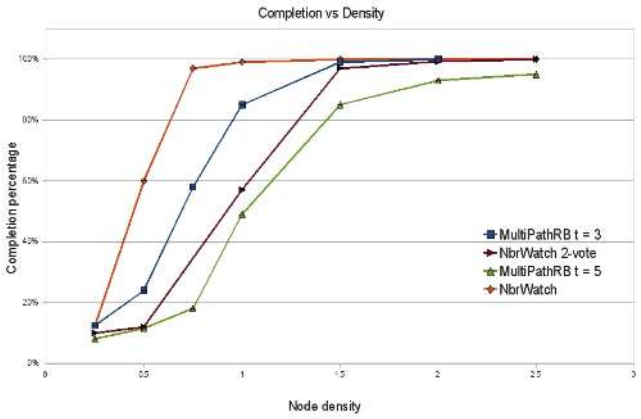

Figure 5: Tolerating crashed devices. Percentage of devices that complete the protocol versus the density of the deployment, for different versions of the protocols. Observe that with a density of 1.5, there are sufficiently many correct nodes for all but MultiPathRB with $t = 5$ to complete almost all the time. In the latter case, the nodes near the edges of the network do not have enough correct neighbors. The experiments were conducted on a 24×24 map.

6.1 Resilience

Resilience to Crash Failures. The first set of experiments, depicted in Figure 5, assumes that nodes fail by crashing, i.e., taking no steps. Effectively, this results in varying numbers of devices that remain active.

For the NeighborWatchRB protocol, broadcasts complete as long as the network remains connected. The 2-Vote version of NeighborWatchRB requires slightly stronger connectivity guarantees, as every bit has to be received from two neighbors. The MultiPathRB protocol requires even stronger connectivity guarantees, as message bits have to be received across $t + 1$ node-disjoint paths.

For each protocol, we varied the number of active nodes, i.e. the density of the network, and examined the percentage of devices that completed the protocol. As the density increases, we observe that almost every device completes the protocol. For MultiPathRB with $t = 5$, however, the devices in the corners of the network do not always receive the broadcast, as they do not always have enough active neighbors. As expected, NeighborWatchRB yields the best results for low densities.

Resilience to Jamming. The second set of experiments examines performance in the presence of jamming. These experiments were run with 800 devices on a 24×24 map (i.e., a density of ~ 1.5), where 10% were selected at random to jam. Each malicious device broadcasts a jamming message in each veto round with probability 1/5. (We found this probability to be approximately optimal for the jammers, as it prevented too much redundant jamming.) During the experiment, we varied the budget of broadcasts allocated to each malicious device, and observed how long it took for the algorithm to complete. Despite the jamming, the protocols complete much as expected. There is a linear relationship between the amount of jamming and the delay, i.e., damage caused by the Byzantine devices is proportional to the amount of jamming. The graph is omitted due to space constraints.

Resilience to Lying. In the third set of experiments, we study the resilience to lying, that is, to Byzantine devices attempting to persuade honest devices to adopt an incorrect value. The experiments were performed on a 20×20 map with range $R = 4$ and 600 nodes (i.e., a density of ~ 1.5).

We simulate the Byzantine nodes by initializing the "corrupt" devices with a fake message to propagate, but have them run the *correct* protocol. These devices appear correct, hence their neighbors are likely to adopt the fake message. For MultiPathRB, the corrupt devices broadcast COMMIT messages for the fake value, and they never relay HEARD messages from correct nodes. For NeighborWatchRB, the malicious devices act as sources initialized with the fake message. Throughout these experiments, we do not limit the broadcast budget of the malicious devices.

Figure 6 presents the relation between the percentage of Byzantine devices and the percentage of nodes that receive the correct message. For MultiPathRB, observe that the ability to tolerate malicious devices increases with the tolerance threshold t, as expected: for $t = 3$, the theoretic analysis implies a tolerance of approximately 2.5%, and for $t = 5$, a tolerance of approximately 5% (Each device has approximately 80 neighbors, in expectation, and these numbers represent 3/80 and 5/80, respectively.). We can readily prevent devices from ever delivering a fake message by increasing the threshold t; however, in that case, some devices never deliver any message. For example, for $t = 9$ (not shown in Figure 6), even for 15% corrupt devices, almost no honest device delivers a fake message. However, only 15% of devices deliver any message at all!

The NeighborWatchRB protocol tolerates larger densities of Byzantine nodes in practice, despite the worst-case theoretical analysis. In fact, the probability of success depends only on the probability that in any square containing a cor-

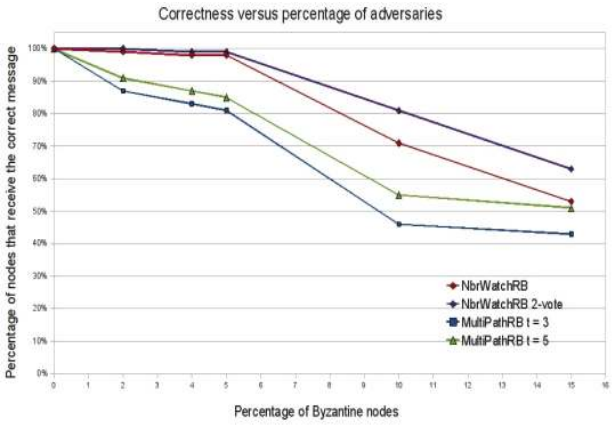

Figure 6: Tolerating lying devices. The percentage of delivered messages that are correct, versus the percentage of malicious devices for different variants of the protocols. NeighborWatchRB performs better than MultiPathRB, even though, in theory, it should be less resilient. The experiments were performed on a 20×20 map, with 600 nodes.

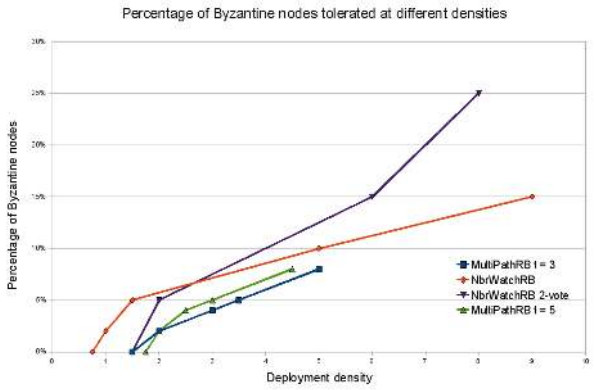

Figure 7: Tolerating lying devices. For a given deployment density, the graph shows the maximum percentage of Byzantine nodes tolerated in order for at least 90% of honest nodes to receive the correct message, for different versions of the protocols. NeighborWatchRB benefits most from the increase in density. Experiments are performed on a 20×20 map, with 300 to 3600 nodes.

rupt device, there is also an honest device. The two-voting variant is more robust, as a corrupt message is only propagated if two nearby squares are populated only by corrupt devices.

Notice that, for both protocols, there is a steep drop-off after the threshold of tolerated Byzantine devices is exceeded. This is because of a snowball effect: once honest devices start adopting the incorrect message, the process accelerates as they join in to convince their neighbors to adopt the incorrect message. This effect is accentuated for lower deployment densities.

A secondary goal is that robustness should scale well with density: as we deploy more devices, we should achieve a higher level of robustness. In Figure 7, we examine this relationship. (Our experiments involving MultiPathRB max out at a density of 5, as the simulation becomes prohibitively slow.) At high levels of density, NeighborWatchRB can tolerate up to 25% of the devices being corrupted. In this sense, the robustness of NeighborWatchRB scales well with density.

6.2 Additional Experiments

Non-uniform Node Distributions. We have also run a series of experiments for NeighborWatchRB where the devices are not distributed uniformly at random, but are deployed in clusters. More specifically, we choose at random a fixed set of *cluster centers*; each device is randomly assigned to a cluster, and within a cluster, devices are spread according to a normal distribution. (The algorithm used for generating the normal distribution of points is that of Marsaglia [21].) The experiments were performed for $R = 4$ and 1200 nodes on a 30×30 map.

We found that, as long as there is sufficient connectivity, the NeighborWatchRB algorithm continues to work well. The algorithm does not always attain 100% completion, since a small fraction of the nodes are disconnected from the source. In experiments where a fraction of the nodes are Byzantine, NeighborWatchRB benefits from the inherent clustering of the nodes, which increases its correctness ratio by up to 10%, when compared to the uniform distribution.

Varying Map Size. We have also verified the performance of NeighborWatchRB on maps of varying sizes. We have found that both the running time and message complexity scale linearly with the diameter of the network, much as expected.

Comparison with simple Epidemic algorithm. As a point of comparison, we implemented a simple epidemic protocol that provides no resilience to faults or jamming. Unsurprisingly, such an algorithm is more efficient than any of our fault-tolerant protocols. The epidemic algorithm is orders of magnitude faster than MultiPathRB. This considerable gap in terms of performance can be explained by the complexity of the voting protocol that MultiPathRB employs, combined with the fact that messages are sent bit by bit. By contrast, NeighborWatchRB performs a lot better, taking on average about 7.7 times longer to complete. The experiments were performed on maps from 30×30 to 50×50, with a node density of 1.25, a range of 3, and message length of 5 bits. Each experiment was repeated 20 times. Much of the reason for the good performance comes from the fact that data propagation is pipelined, so that the additional cost incurred by NeighborWatchRB can be amortized against the inherent cost of sending data across a multi-hop network.

These experiments support our conjecture that a dual-mode protocol combining an epidemic broadcast of the entire message, along with a secure broadcast of short digest, may be sufficiently efficient. In particular, it seems plausible that a sufficient level of security can be achieved with a digest that is 1/10 the size of the original message, which would yield a slow down of less than a factor of 2 for providing Byzantine fault-tolerance.

7. CONCLUDING REMARKS

We have presented two authenticated broadcast protocols for multi-hop wireless networks subject to Byzantine failures. Together, these two protocols take a first step toward

better understanding the problem of authenticated broadcast. In terms of future work, several improvements might be considered, such as improving message efficiency, weakening the synchronization requirements, and adapting the protocol to mobile nodes.

Acknowledgements. The authors would like to thank Prof. Guevara Noubir for his useful comments on earlier drafts of this paper.

8. REFERENCES

[1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *Operating Systems Review*, 32:9–20, 1998.

[2] B. Awerbuch, A. Richa, and C. Scheideler. A jamming-resistant mac protocol for single-hop wireless networks. In *PODC '08*.

[3] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. On the performance of ieee 802.11 under jamming. *INFOCOM 2008*.

[4] V. Bhandari and N. Vaidya. On reliable broadcast in a radio network. In *PODC '05*, pages 138–147.

[5] V. Bhandari and N. Vaidya. On reliable broadcast in a radio network: A simplified characterization. Technical report, University of Illinois at Urbana-Champaign, May 2005.

[6] V. Bhandari and N. Vaidya. Reliable broadcast in a wireless grid network with probabilistic failures. Technical report, University of Illinois at Urbana-Champaign, February 2006.

[7] M. Blaum and H. van Tilborg. On t-error correcting/all unidirectional error detecting codes. *IEEE Trans. Computers*, 38(11):1493–1501, 1989.

[8] Bluetooth Consortium. *Bluetooth Specification v. 2.1*, July 2007.

[9] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *EUROCRYPT*, 2001.

[10] J. M. Borden. Optimal asymmetric error detecting codes. *Information and Control*, 53(1/2):66–73, 1982.

[11] M. Cagalj, S. Capkun, R. Rengaswamy, I. Tsigkogiannis, M. Srivastava, and J.-P. Hubaux. Integrity (i) codes: Message integrity protection and authentication over insecure channels. In *IEEE Security and Privacy*, 2006.

[12] S. Capkun, M. Cagalj, R. Rengaswamy, I. Tsigkogiannis, J.-P. Hubaux, and M. Srivastava. Integrity codes: Message integrity protection and authentication over insecure channels. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):208–223, Oct.-Dec. 2008.

[13] S. Capkun and J.-P. Hubaux. Secure positioning of wireless devices with application to sensor networks. *INFOCOM '05*, pages 1917–1928.

[14] A. Chan, X. Liu, G. Noubir, and B. Thapa. Broadcast control channel jamming: Resilience and identification of traitors. *ISIT '07*.

[15] J. Chiang and Y.-C. Hu. Cross-layer jamming detection and mitigation in wireless broadcast networks. In *MobiCom '07*, pages 346–349. ACM, 2007.

[16] S. Dolev, S. Gilbert, R. Guerraoui, and C. Newport. Secure communication over radio channels. In *PODC '08*.

[17] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Des. Test*, 24(6):522–533, 2007.

[18] Full version of this paper. http://lpd.epfl.ch/alistarh/wireless-byzantine/.

[19] S. Gilbert, R. Guerraoui, and C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. In *OPODIS '06*.

[20] S. Gilbert, R. Guerraoui, and C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. *Theoretical Computer Science*, 2008.

[21] D. Knuth. *The Art of Computer Programming, Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1997.

[22] C-Y. Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *PODC '04*.

[23] C-Y. Koo, V. Bhandari, J. Katz, and N. Vaidya. Reliable broadcast in radio networks: The bounded collision case. In *PODC '06*.

[24] Xin Liu, G. Noubir, R. Sundaram, and San Tan. Spread: Foiling smart jammers using multi-layer agility. *INFOCOM 2007*, May 2007.

[25] V. Navda, A. Bohra, S. Ganguly, and D. Rubenstein. Using channel hopping to increase 802.11 resilience to jamming attacks. *INFOCOM 2007*.

[26] G. Noubir and G. Lin. On link layer denial of service in data wireless lans. *Wiley Journal on Wireless Comm. and Mobile Computing*, August 2005.

[27] A. Pelc and D. Peleg. Feasibility and complexity of broadcasting with random transmission failures. In *PODC*, 2005.

[28] A. Perrig, J. D. Tygar, D. Song, and R. Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *SP '00*, page 56, 2000.

[29] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Mobile-assisted localization in wireless sensor networks. In *INFOCOM '05*, pages 172–183.

[30] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *MOBICOM*, 2000.

[31] M. Strasser, C. Pöpper, S. Capkun, and M. Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *IEEE SP '08*.

[32] WSNet / Worldsens simulator: an event-driven simulator for large-scale wireless networks. http://wsnet.gforge.inria.fr/.